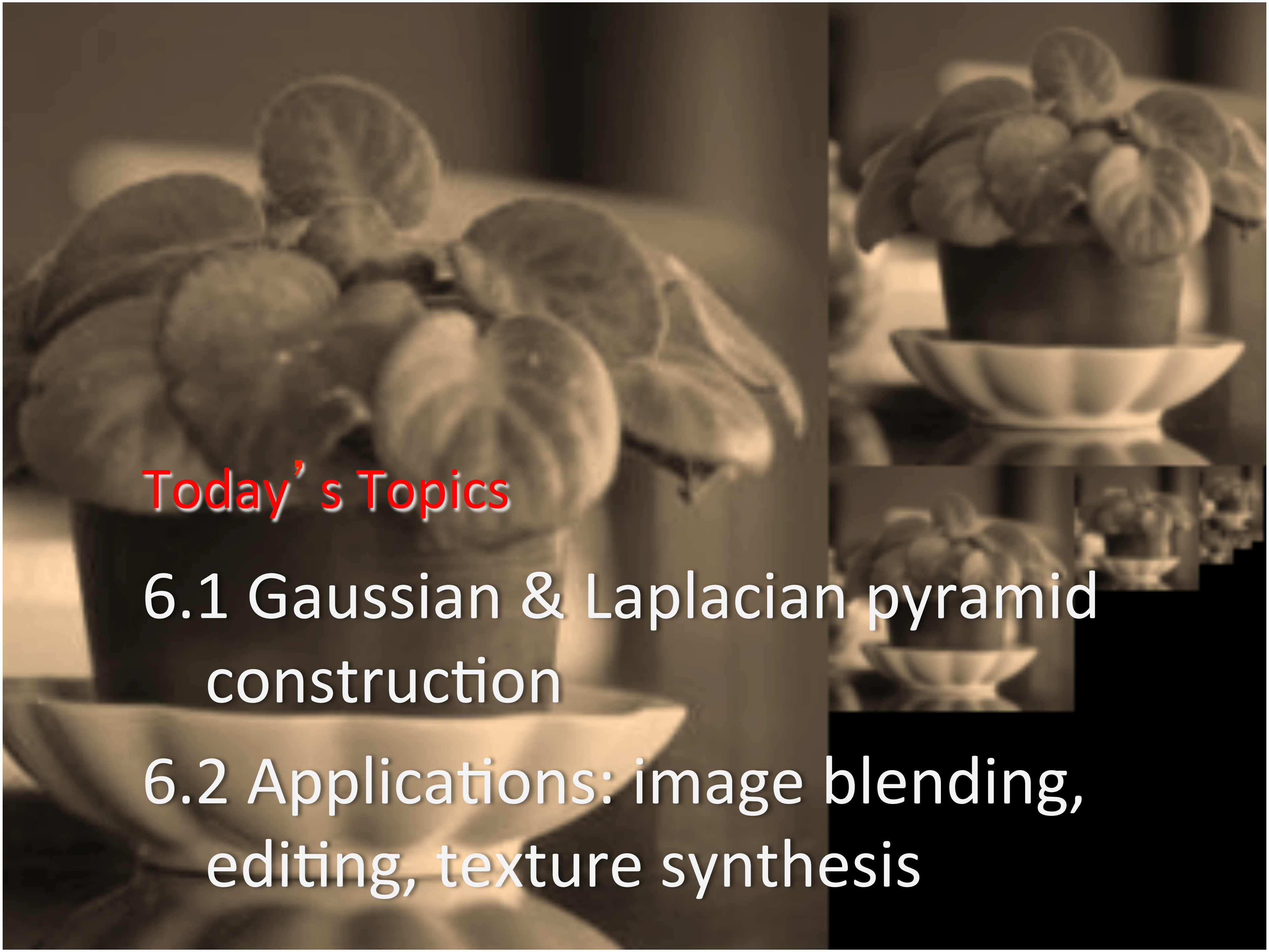


Today's Topics

6.1 Gaussian & Laplacian pyramid construction

6.2 Applications: image blending, editing, texture synthesis

Topic 6:

Hierarchical image representations

1. Gaussian & Laplacian pyramids
2. Applications:
    1. Multi-resolution image blending
    2. Multi-resolution image editing
    3. Multi-resolution texture synthesis

Topic 6.1:

Gaussian & Laplacian Pyramids

- The Gaussian pyramid (intro)
- The convolution operation
- Constructing the gaussian pyramid
    - The REDUCE() function
- Constructing the Laplacian pyramid
    - The EXPAND() function

The Gaussian Pyramid

The Gaussian Pyramid: A representation in multiple scales

Original Image

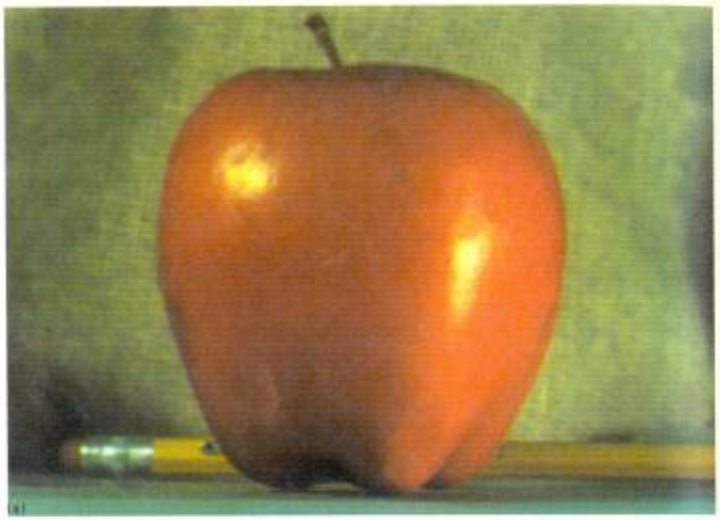
The Gaussian Pyramid

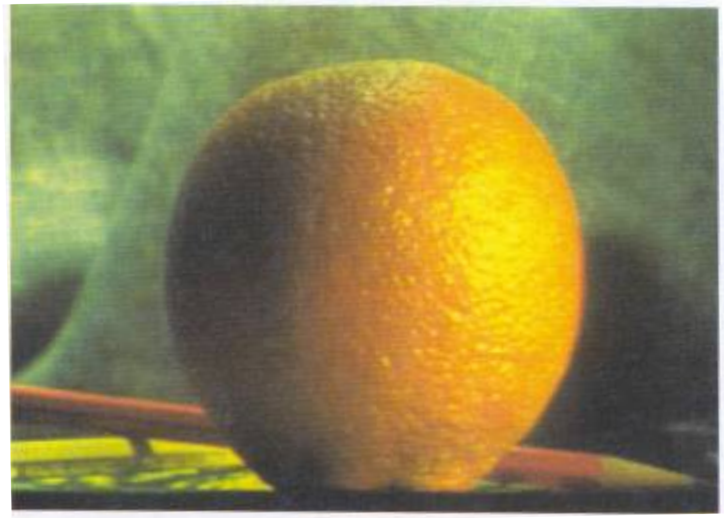
The goal is to define a representation in which image information at different scales is explicitly available (i.e. does not need to be computed when needed)

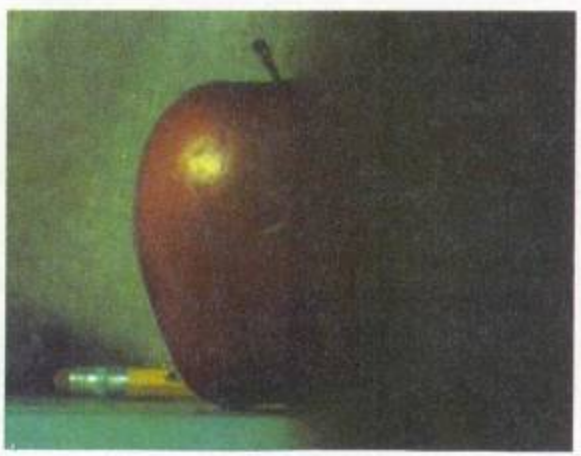
Applications:
- Scale invariant template matching (like faces)
- Progressive image transmission
- Image blending
- Efficient feature search
- …

Application 1: Pyramid Image Blending

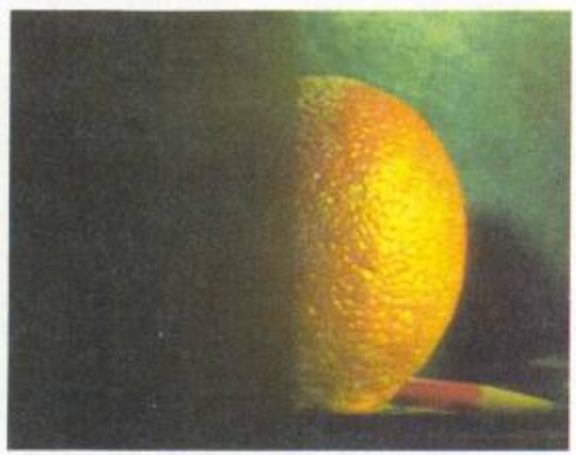
Goal: Merge two images without visible seams



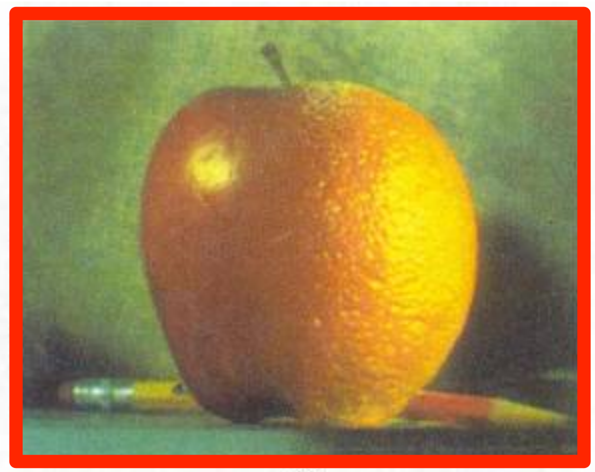




(d)

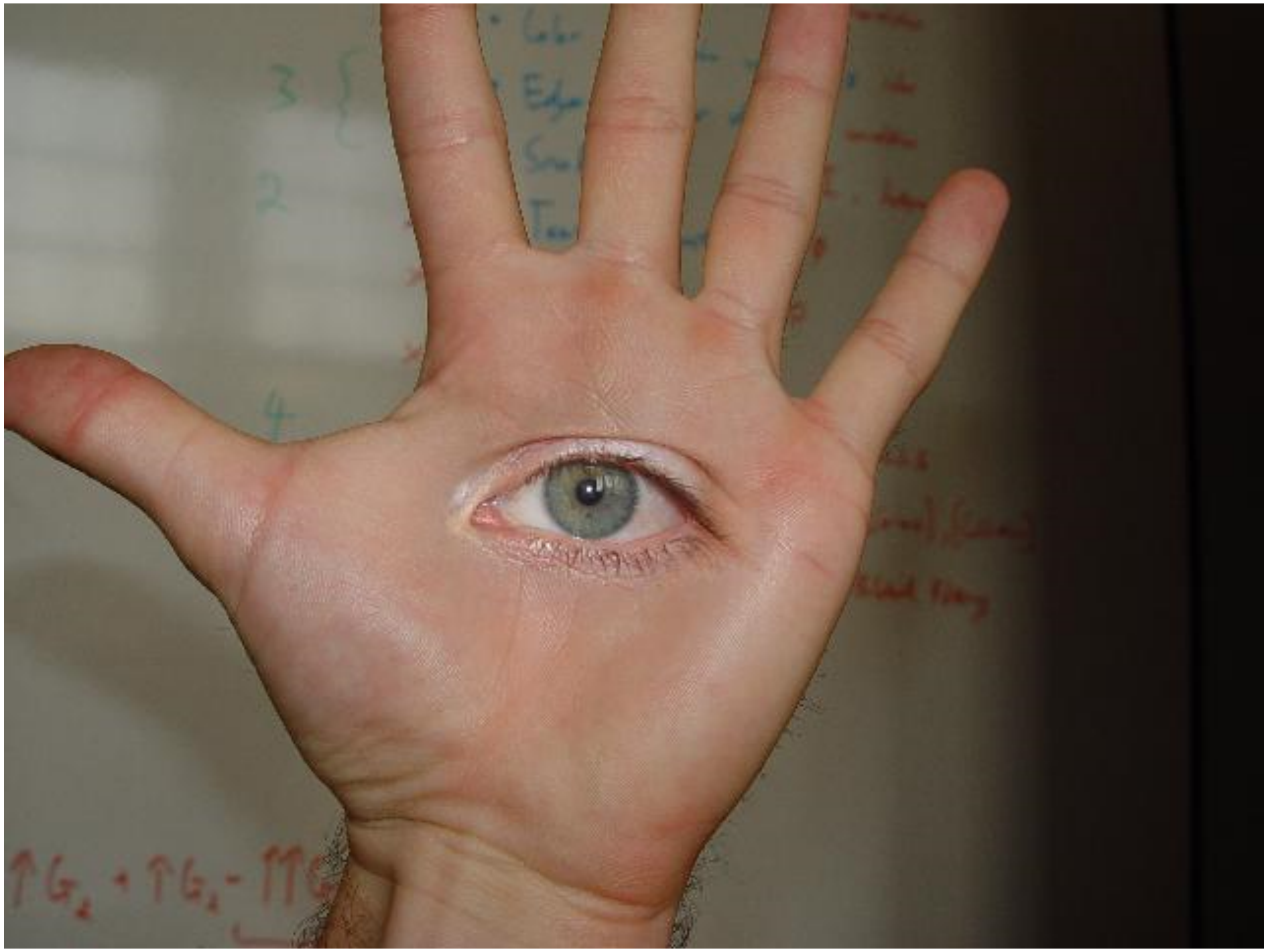


(h)



(l)

Horror Photo

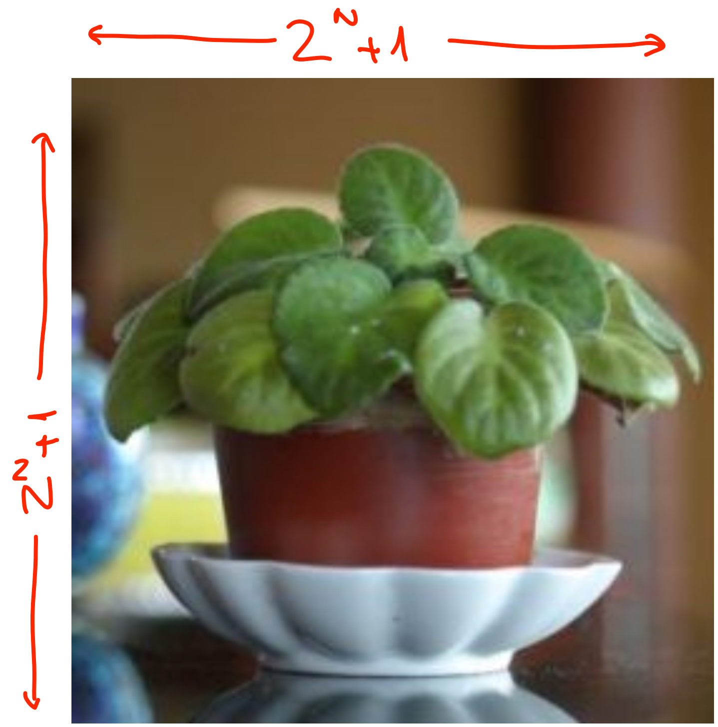


© prof. dmartin

The Gaussian Pyramid

The elements of a Gaussian Pyramids are smoothed copies of the image at different scales.

Input: Image I of size $(2^N+1)x(2^N+1)$



The Gaussian Pyramid

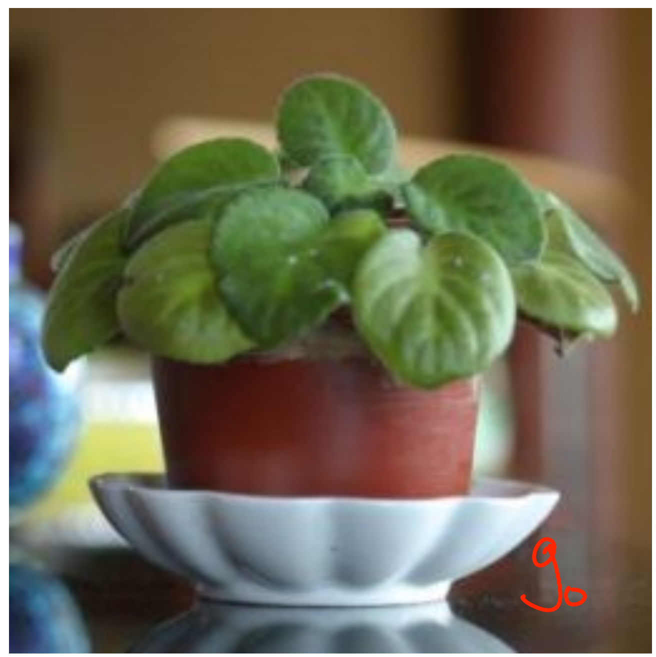
The elements of a Gaussian Pyramids are smoothed copies of the image at different scales.

Input: Image I of size $(2^N+1)x(2^N+1)$

Output: Images g_0

2^N+1

2^N+1



g_0

Note: The original image is part of the output!

The Gaussian Pyramid

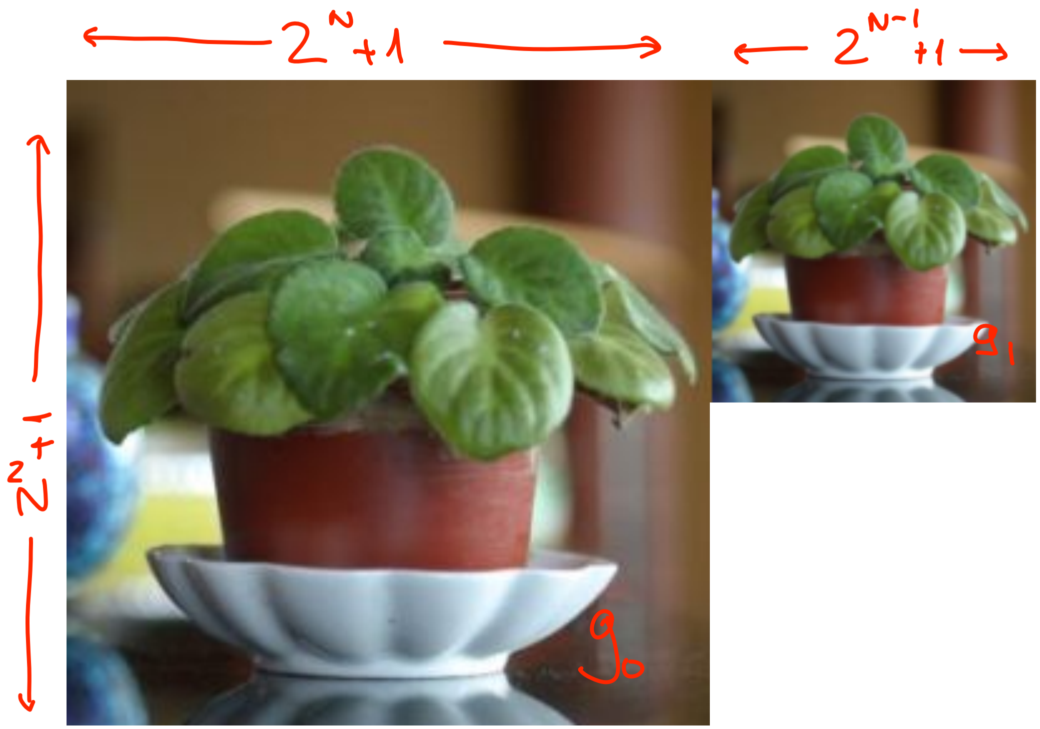
The elements of a Gaussian Pyramids are smoothed copies of the image at different scales.

Input: Image I of size $(2^N+1)x(2^N+1)$

Output: Images g_0, g_1

where the size of g_1 is:

$$(2^{N-1}+1)x(2^{N-1}+1)$$



The Gaussian Pyramid

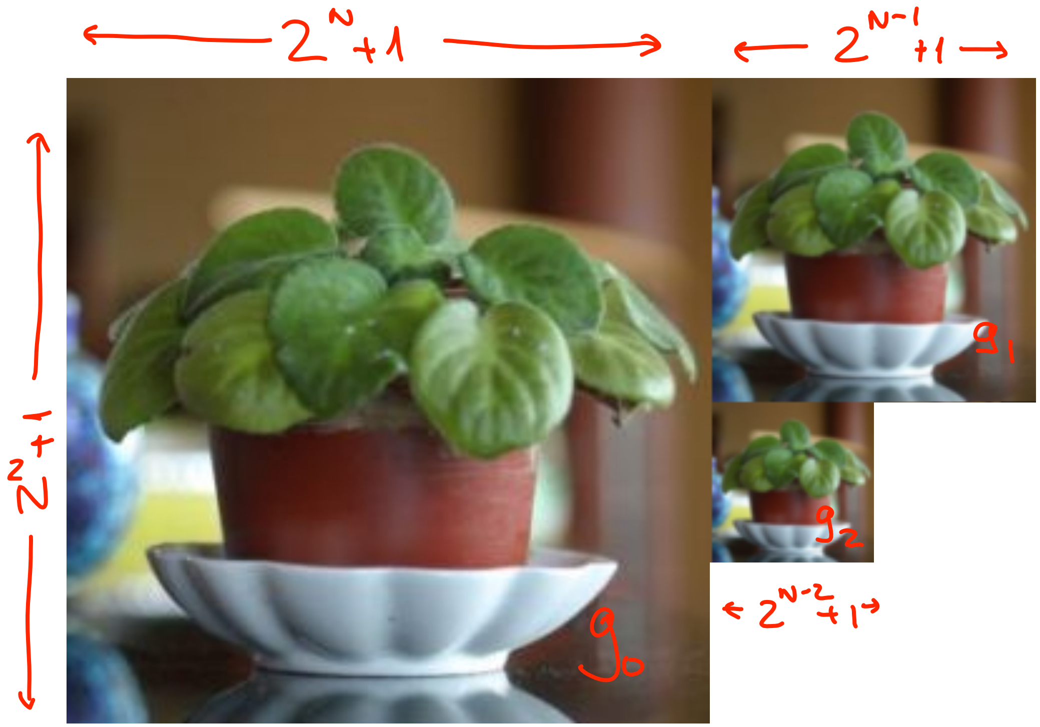
The elements of a Gaussian Pyramids are smoothed copies of the image at different scales.

Input: Image I of size $(2^N+1)x(2^N+1)$

Output: Images g_0, g_1, g_2

where the size of g_2 is:

$$(2^{N-2}+1)x(2^{N-2}+1)$$



The Gaussian Pyramid

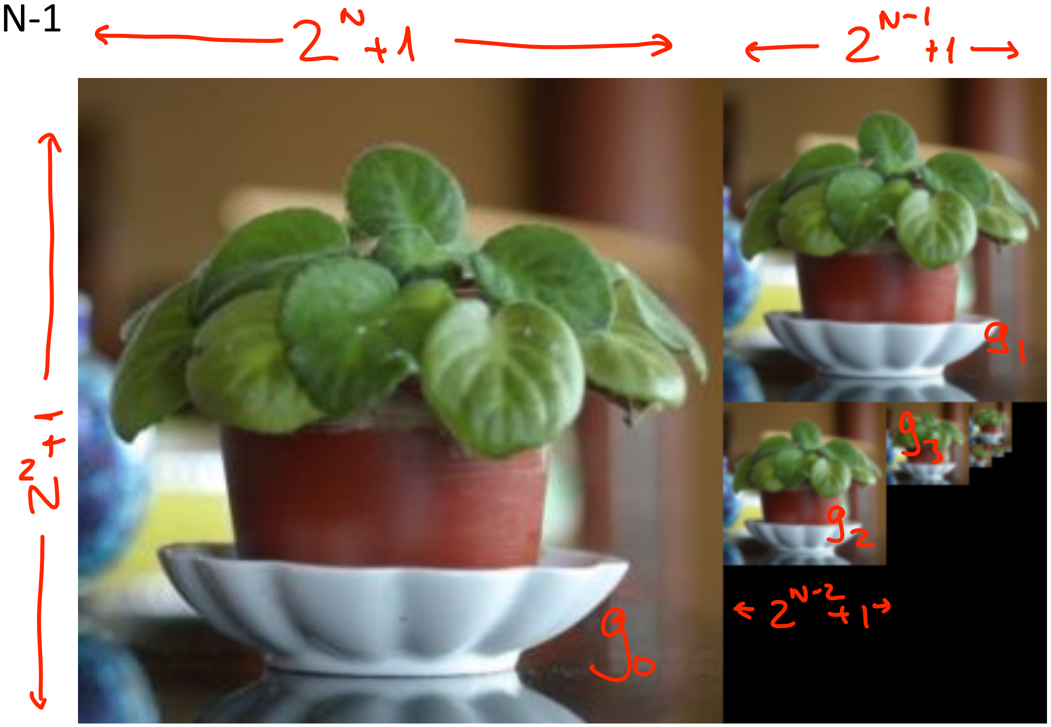
The elements of a Gaussian Pyramids are smoothed copies of the image at different scales.

Input: Image I of size $(2^N+1)x(2^N+1)$

Output: Images $g_0, g_1, \ldots, g_{N-1}$

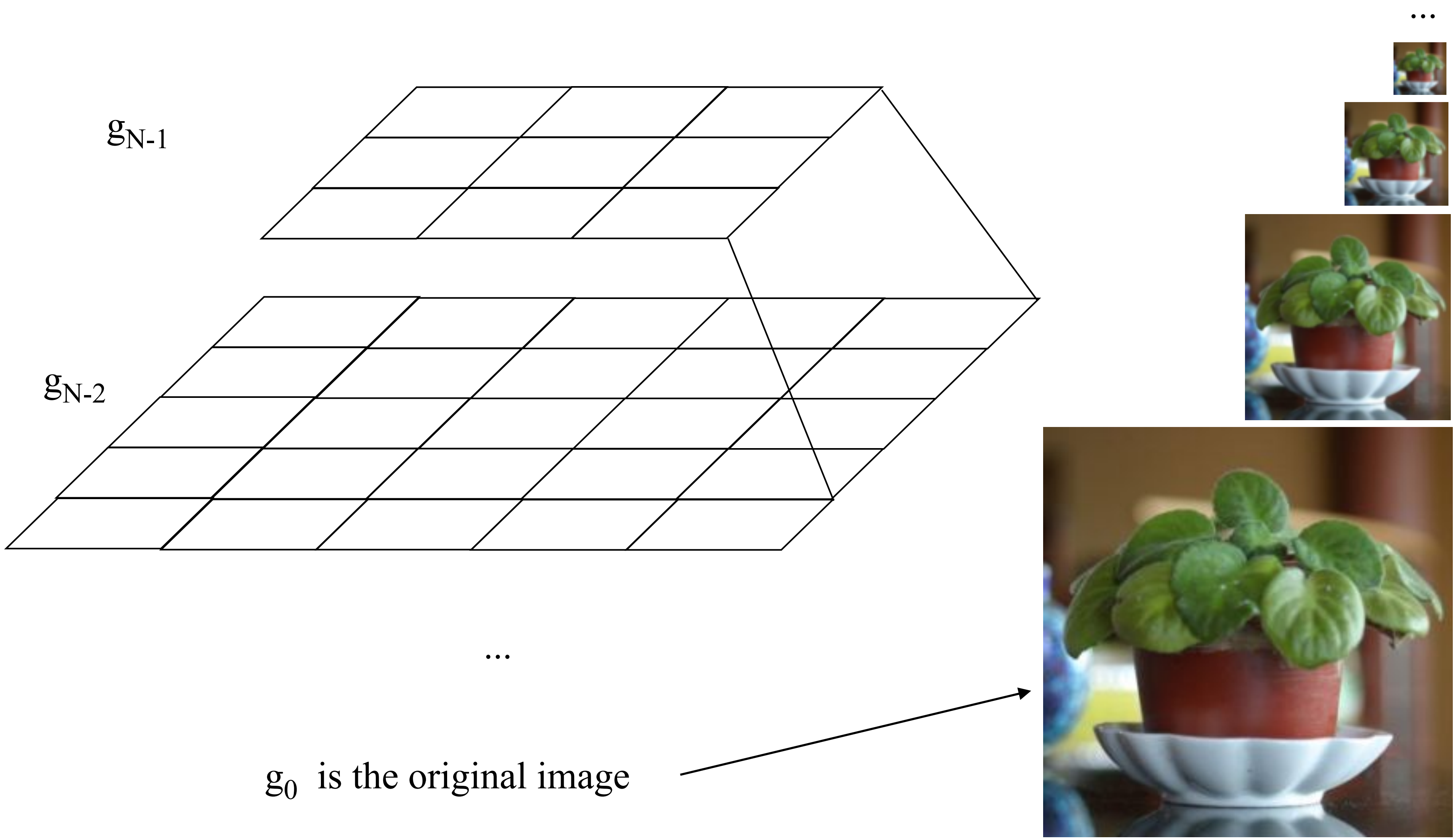
where the size of g_i is:

$$(2^{N-i}+1)x(2^{N-i}+1)$$

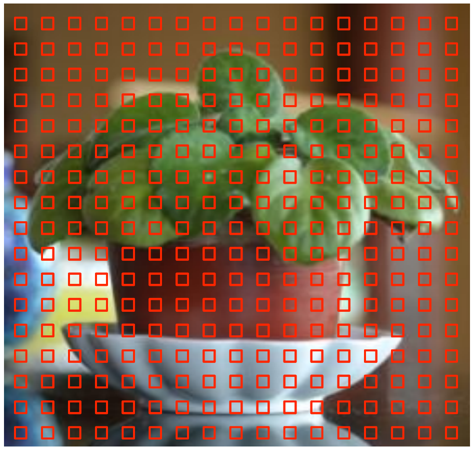


And they called this a Pyramid?

Yes, because the representation can be pictured as a pyramid of 3×3, 5×5, 9×9,…, $(2^N+1) \times (2^N+1)$ images when stacked.

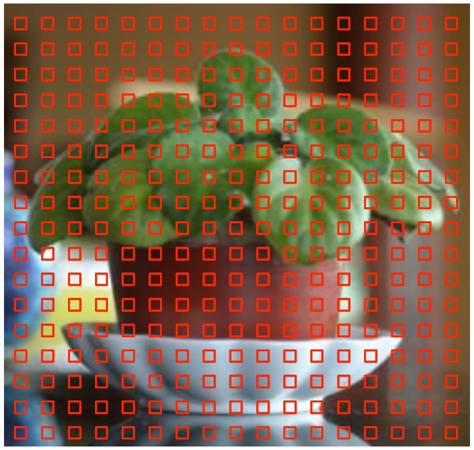


This is silly, why not just sub-sample?

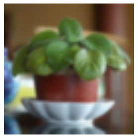


Take every 2^{nd} pixel from I for g_1

This is silly, why not just sub-sample?

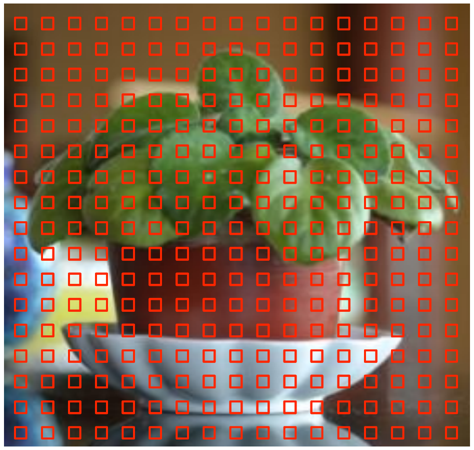


Take every 2^{nd} pixel from I for g_1

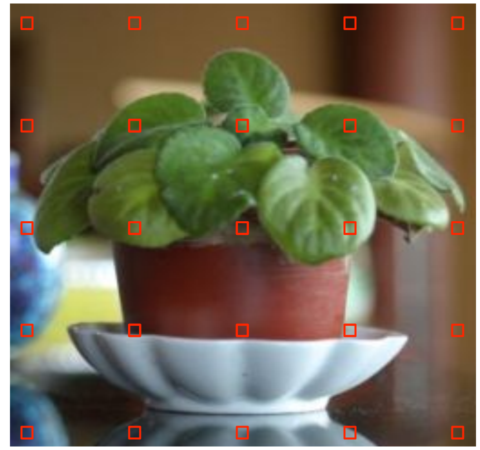


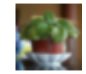
Take every 4^{th} pixel from I for g_2

This is silly, why not just sub-sample?

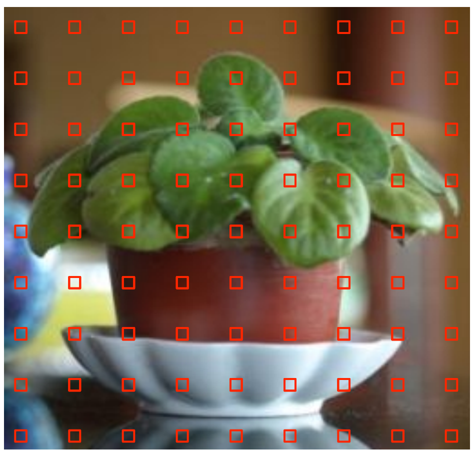


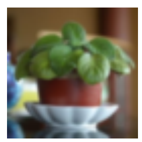
Take every 2nd pixel from I for g_1





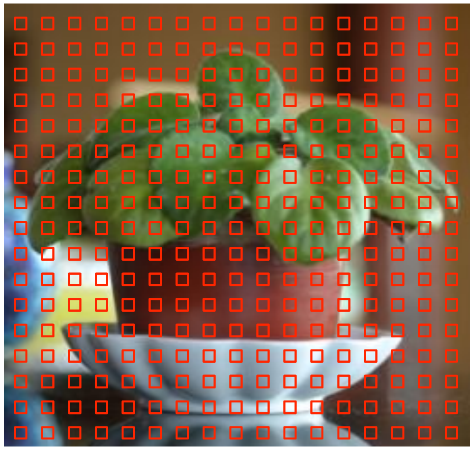
Take every 8th pixel from I for g_3



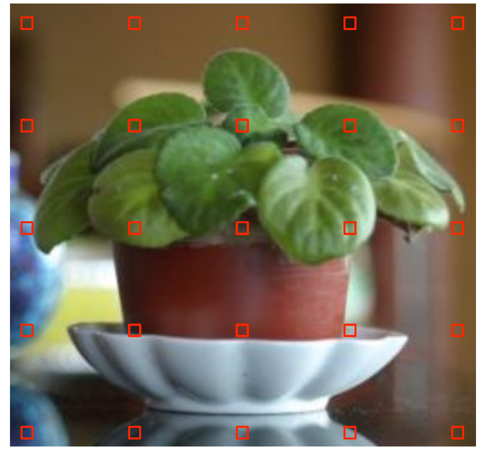


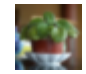
Take every 4th pixel from I for g_2

This is silly, why not just sub-sample?

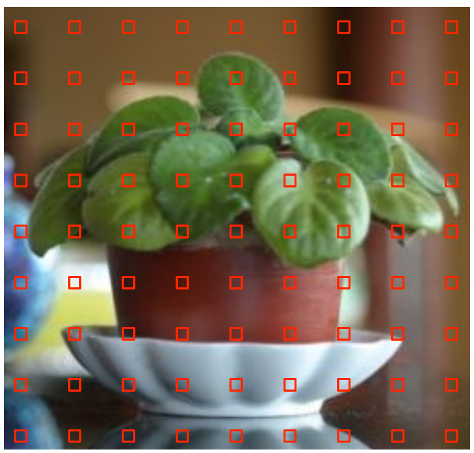


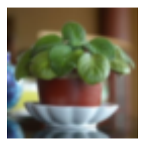
Take every 2nd
pixel from I c



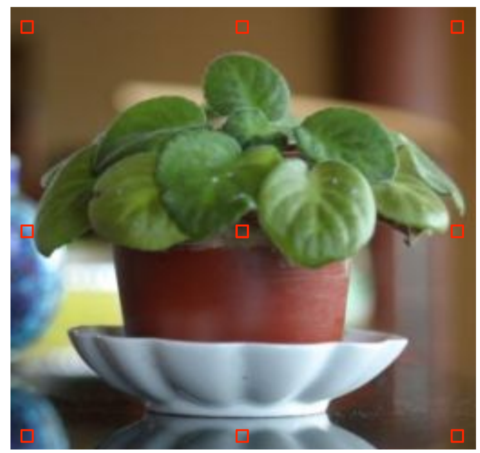


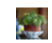
Take every 8th
pixel from I for g_3





Take every 4th
pixel from I for g_2

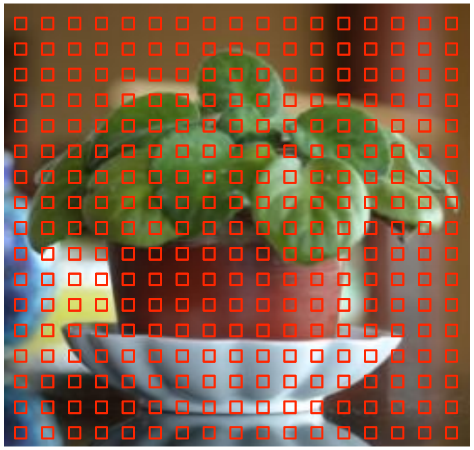




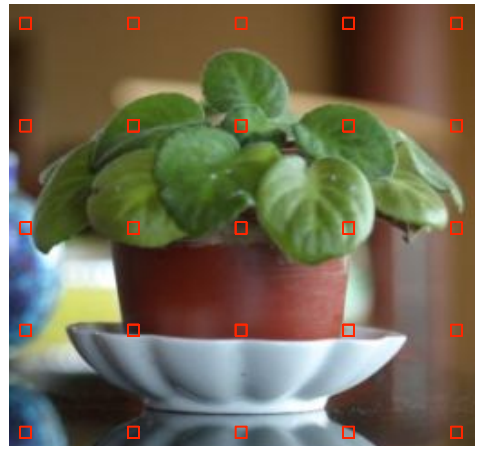
Take every 16th
pixel from I for g_4

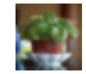
and so on...

This is silly, why not just sub-sample?

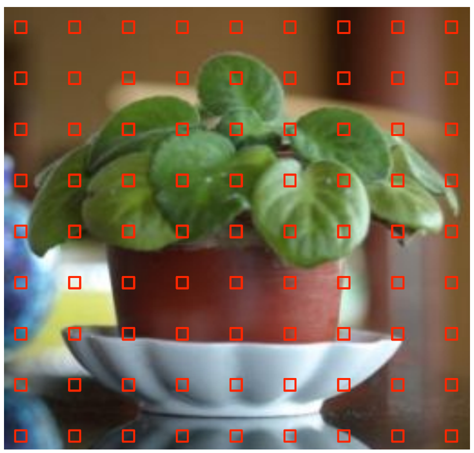


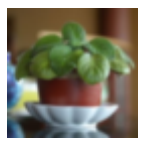
Take every 2^{nd} pixel from I for g_1



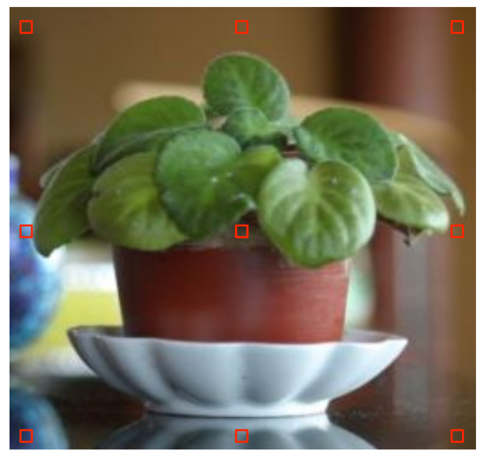


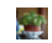
Take every 8^{th} pixel from I for g_3





Take every 4^{th} pixel from I for g_2

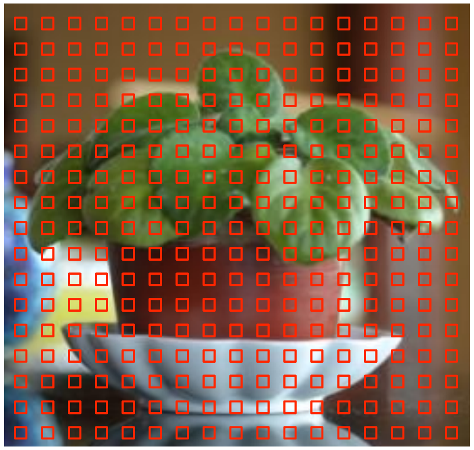




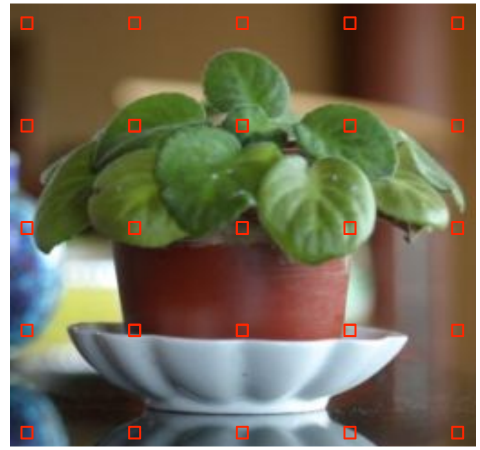
Take every 16^{th} pixel from I for g_4

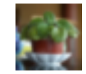
All the information is there already, or is it not?

This is silly, why not just sub-sample?

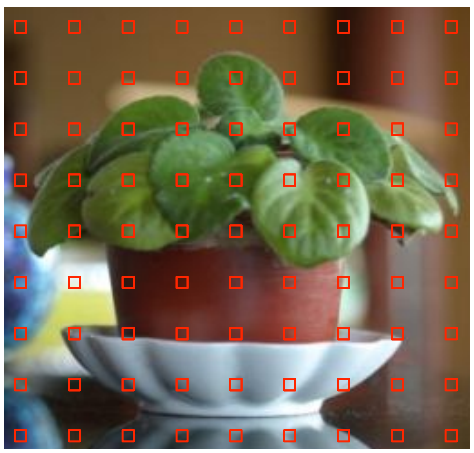


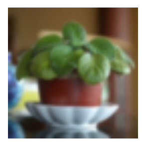
Take every 2^{nd} pixel from I for g_1



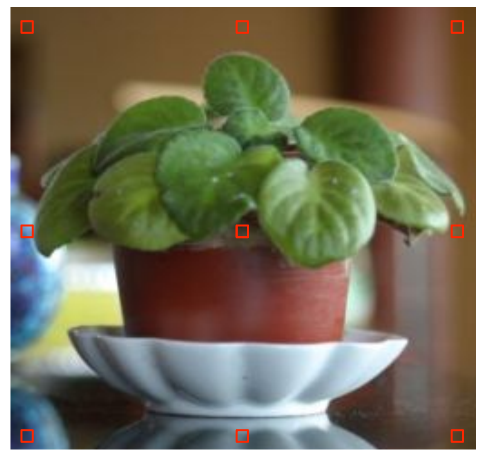


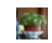
Take every 8^{th} pixel from I for g_3





Take every 4^{th} pixel from I for g_2

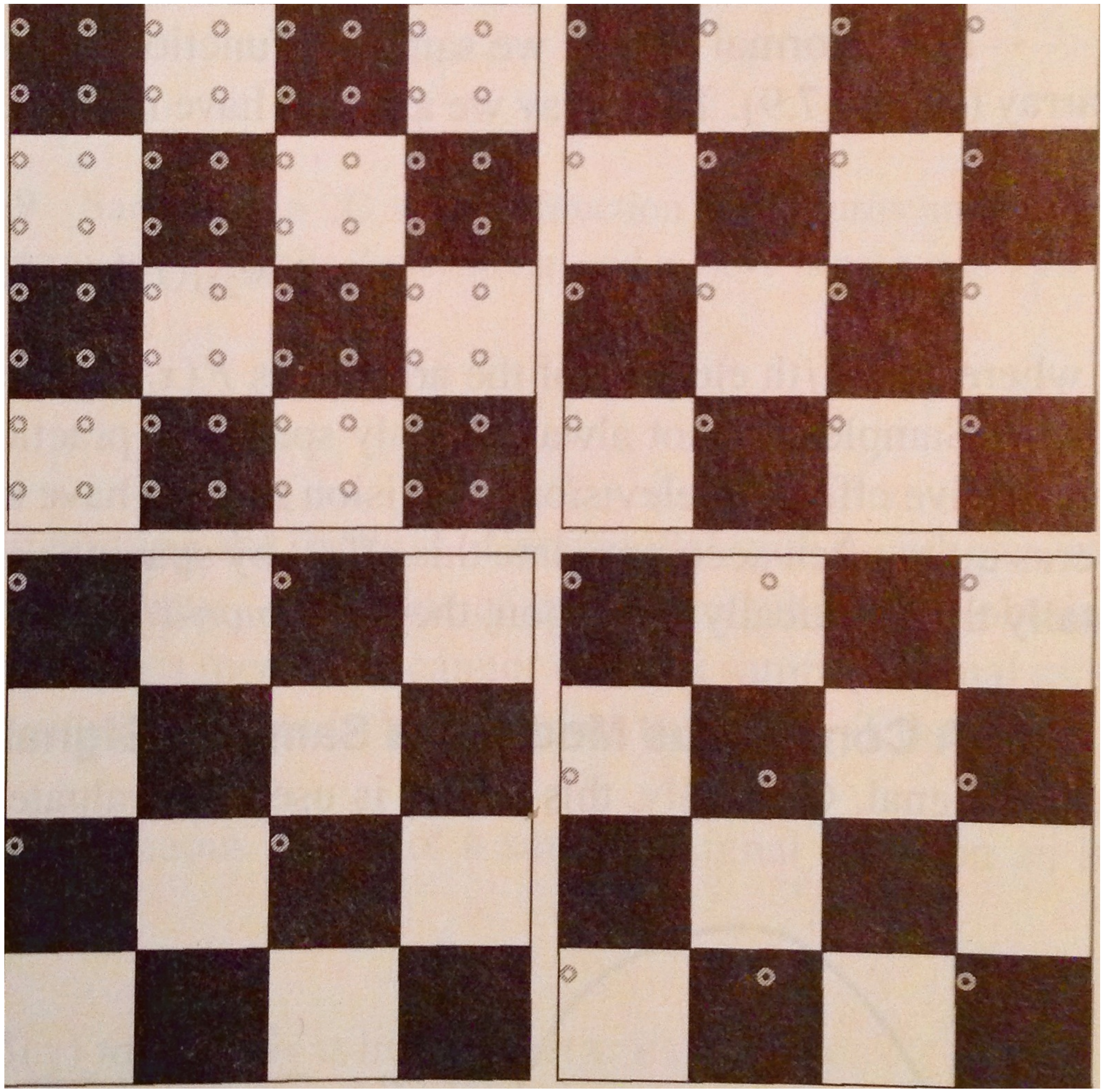




Take every 16^{th} pixel from I for g_4

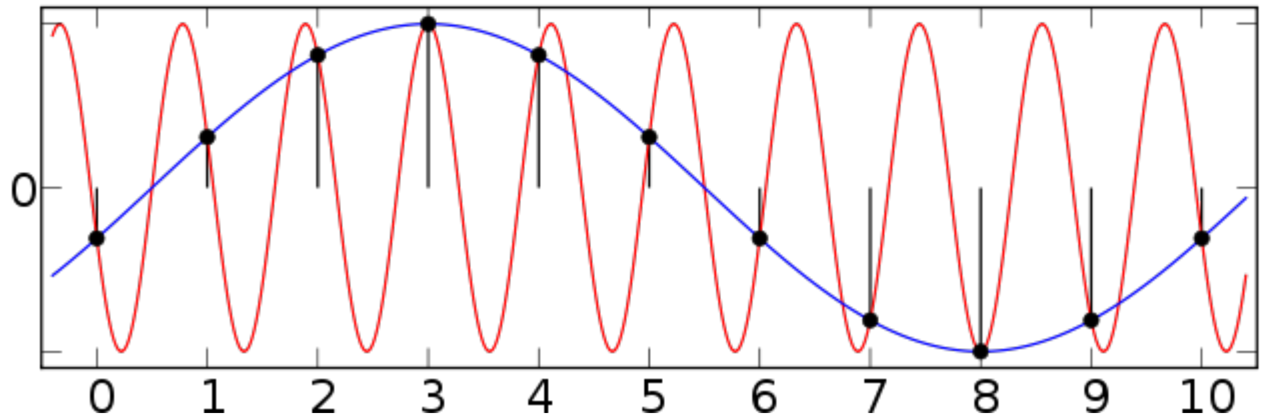
NO! SUBSAMPLING ALONE LEADS TO ALIASING

Aliasing?



Forsyth and Ponce

Sub-sampling is not enough!

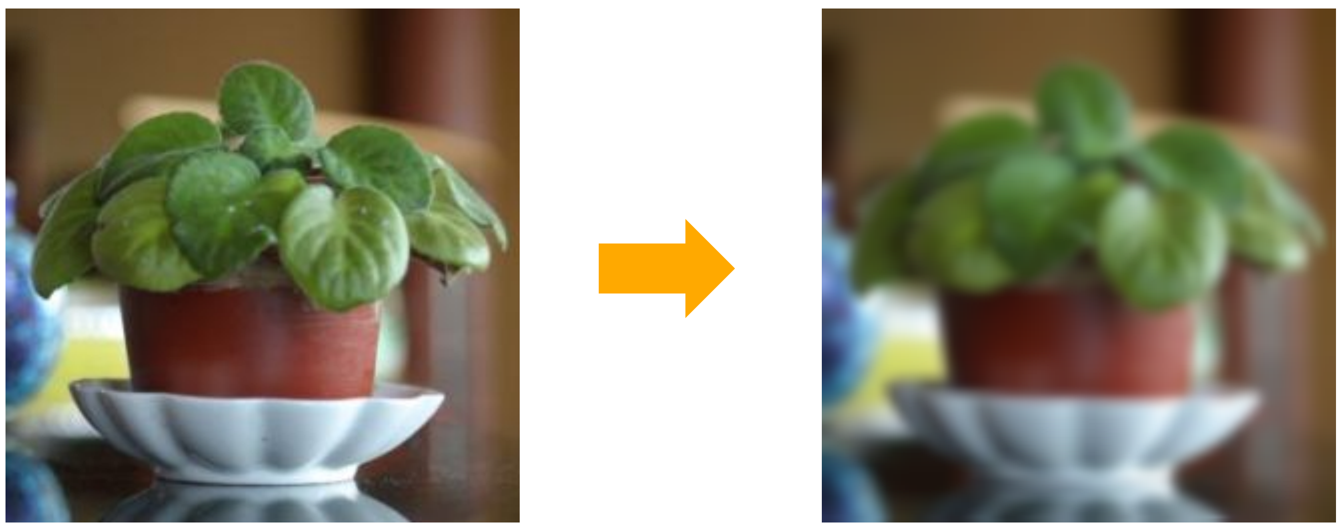


Because of “Aliasing”, sub-sampling may have catastrophic effects.

Aliasing arises when a signal is sampled at a rate that is insufficient to capture the changes in the signal (and in Gaussian Pyramids, this will always happen as sub-sampling is twice as sparse at each level!)

Smoothing

The solution to aliasing effects is smoothing, effectively reducing the maximum frequency of image features.
In other words smoothing removes the fast changes that sub-sampling would miss.



Smoothing

Building (or computing) a Gaussian Pyramid

To generate a Gaussian pyramid, iterate between these two steps:

Smoothing: Remove high-frequency components that could cause aliasing.

Down-sampling: Reduce the image size by ½ at each level.

Building (or computing) a Gaussian Pyramid

To generate a Gaussian pyramid, iterate between these two steps:

Smoothing: Remove high-frequency components that could cause aliasing.

Down-sampling: Reduce the image size by ½ at each level.

Good, but how do we actually implement this?

Topic 6.1:

Gaussian & Laplacian Pyramids

- The Gaussian pyramid (intro)
- The convolution operation
- Constructing the gaussian pyramid
  - The REDUCE() function
- Constructing the Laplacian pyramid
  - The EXPAND() function

Image Smoothing Using Averaging Masks

We know about (both the normalized and un-normalized) the cross correlation operators.

$$CC(X_i, T) = X_i^T \cdot T$$

$$NCC(X_i, T) = \frac{X_i^T \cdot T}{\|X_i\| \cdot \|T\|}$$

What happens if we evaluate them with

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

or

1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25

everywhere in the image?

Image Smoothing Using Averaging Masks

Smoothing can be achieved by averaging neighboring pixels.

The strength of a smoothing operator is proportional to the number of pixels it averages.

Averaging can be computed as the Cross-Correlation of the image with a constant kernel, like:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

or

1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25

Image Smoothing Using Averaging Masks

Original Image

Image Smoothing Using Averaging Masks

Result of Cross-Correlation with 3x3 Mask

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

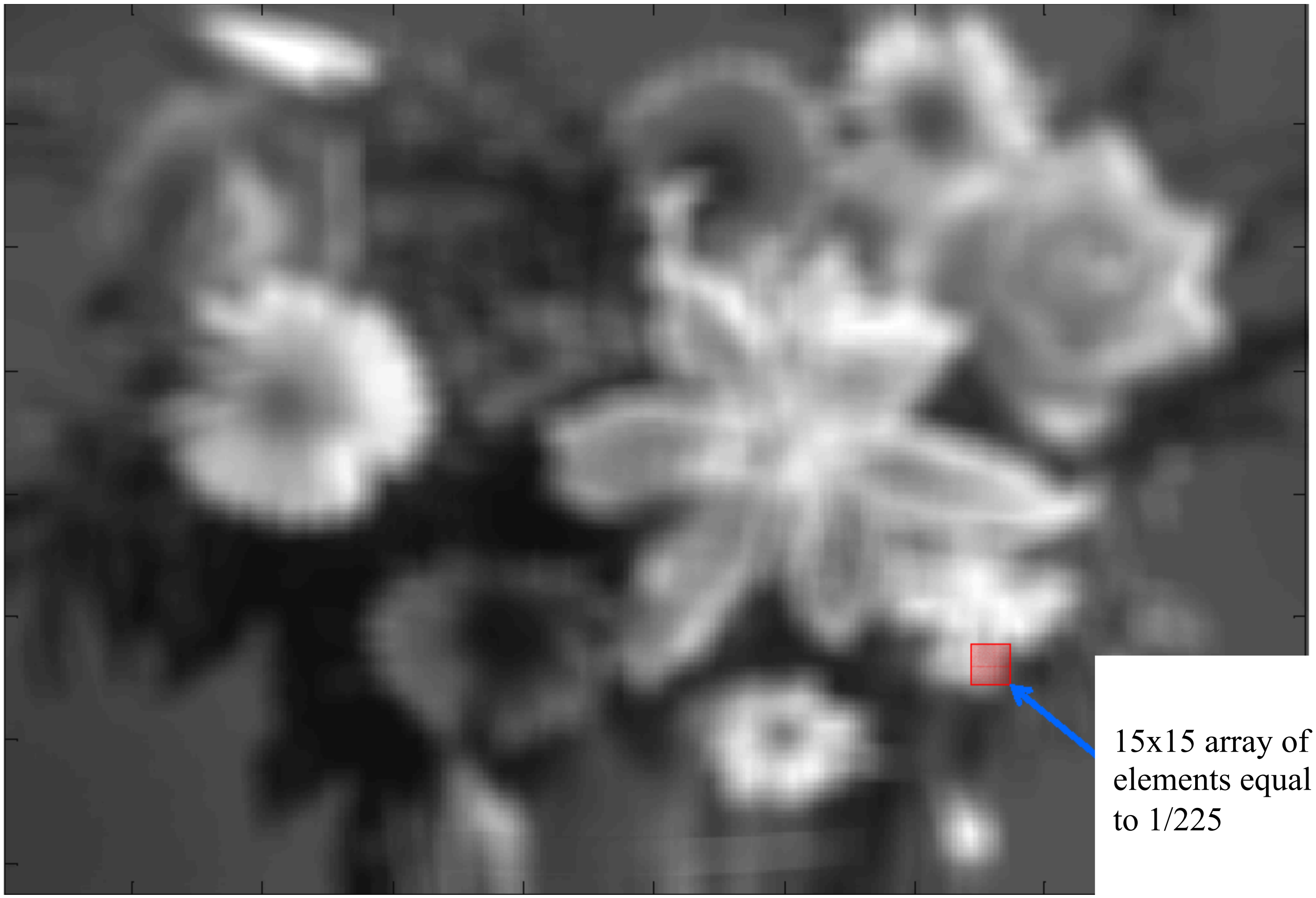
Image Smoothing Using Averaging Masks

Result of Cross-Correlation with 5x5 Mask

1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25

Image Smoothing Using Averaging Masks

Result of Cross-Correlation with 15x15 Mask

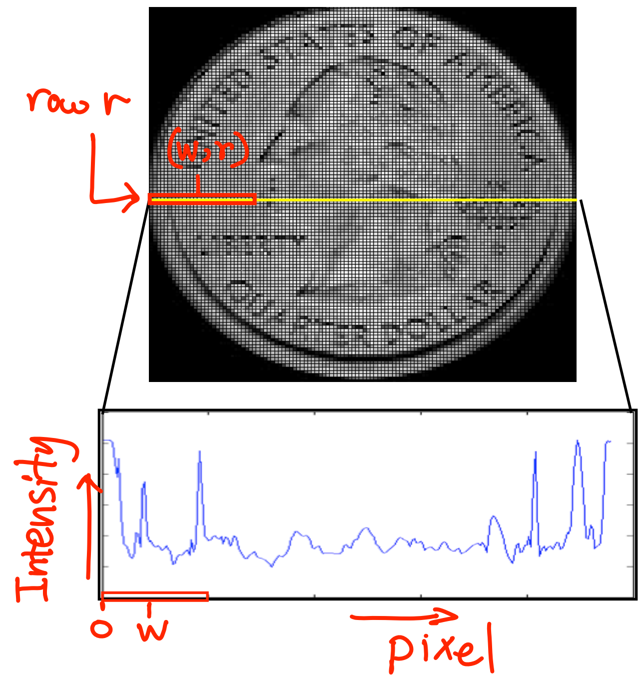


15x15 array of elements equal to 1/225

Image Cross Correlation ⇔ Matrix Multiplication

Cross correlation in 1D can be computed using matrix multiplication, for instance, let:

I: one row of the image (with M pixels)



and

T: a template (with 2w+1 pixels), such as $[T_{-w}, T_{-w-1}, \ldots T_0, \ldots T_{w-1}, T_w]$

Image Cross Correlation ⇔ Matrix Multiplication

Then, the cross correlation at pixel w can be computed using:

I: one row of the image (with M pixels)

T: a template (with 2w+1 pixels), such as $[T_{-w}, T_{-w-1}, \ldots T_0, \ldots T_{w-1}, T_w]$

$$CC(I,T) = \begin{bmatrix} T_{-w} & T_{-w+1} & \cdots & T_0 & \cdots & T_w & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} I_0 \\ I_1 \\ I_2 \\ \vdots \\ I_w \\ \vdots \\ I_{2w} \\ I_{2w+1} \\ \vdots \\ I_{M-1} \end{bmatrix}$$

Image Cross Correlation ⇔ Matrix Multiplication

Then, the cross correlation at pixel w can be computed using:

I: one row of the image (with M pixels)

T: a template (with 2w+1 pixels), such as $[T_{-w}, T_{-w-1}, \ldots T_0, \ldots T_{w-1}, T_w]$

And then the next pixel:

pixel w

$$\begin{bmatrix} T_{-w} & T_{-w+1} & \cdots & T_0 & \cdots & T_w & 0 & \cdots & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & T_{-w} & T_{-w+1} & \cdots & T_{w-1} & T_w & \cdots & 0 \end{bmatrix}$$

pixel w+1

$$\begin{bmatrix} I_0 \\ I_1 \\ I_2 \\ \vdots \\ I_w \\ \vdots \\ I_{2w} \\ I_{2w+1} \\ \vdots \\ I_{M-1} \end{bmatrix}$$

Image Cross Correlation ⇔ Matrix Multiplication

Then, the cross correlation at pixel w can be computed using:

I: one row of the image (with M pixels)

T: a template (with 2w+1 pixels), such as $[T_{-w}, T_{-w-1}, \ldots T_0, \ldots T_{w-1}, T_w]$

And for all the pixels in [w, M-w-1]:

$$\begin{bmatrix} T_{-w} & T_{-w+1} & \cdots & T_0 & \cdots & T_w & 0 & \cdots & 0 \\ 0 & \ddots & & \ddots & & \ddots & 0 & \cdots & 0 \\ & & \ddots & & \ddots & & \ddots & & 0 \\ & 0 & & \ddots & & \ddots & & \ddots & \end{bmatrix} \begin{bmatrix} I_0 \\ I_1 \\ I_2 \\ \vdots \\ I_w \\ \vdots \\ I_{2w} \\ I_{2w+1} \\ \vdots \\ I_{M-1} \end{bmatrix}$$

each row is a 1-pixel right shift of the row above it (called a TOEPLITZ matrix)

Image Cross Correlation ⇔ Matrix Multiplication

Then, the cross correlation at pixel w can be computed using:

I: one row of the image (with M pixels)
T: a template (with 2w+1 pixels), such as $[T_{-w}, T_{-w-1}, \ldots T_0, \ldots T_{w-1}, T_w]$

And for all the pixels in [w, M-w-1]:

$$\begin{bmatrix} T_{-w} & T_{-w+1} & \cdots & T_0 & \cdots & T_w & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & & \ddots & & \ddots & 0 & 0 \\ & & \ddots & \ddots & & \ddots & & \ddots & 0 \\ & 0 & & \ddots & \ddots & & \ddots & & \ddots \end{bmatrix} \begin{bmatrix} I_0 \\ I_1 \\ I_2 \\ \vdots \\ I_w \\ \vdots \\ I_{2w} \\ I_{2w+1} \\ \vdots \\ \\ I_{M-1} \end{bmatrix}$$

each row is a 1-pixel right shift of the row above it
(called a TOEPLITZ matrix)

How about pixels in:

[1, 2, …, w]

and

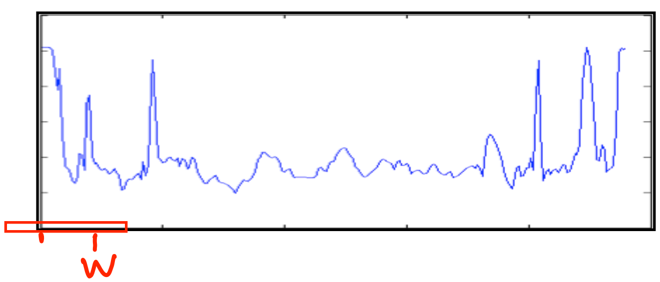
[M-w, M-w+1, …, M]?

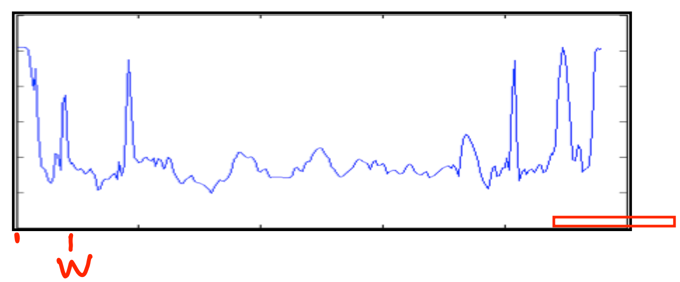
Image Cross Correlation ⇔ Matrix Multiplication

How about pixels at [1, 2, …, w] and [M-w, M-w+1, …, M]?

Options (to taste, with advantages and disadvantages):

- Wrap around in I
- Define new templates
- Assume I is zero for the out-of-image
- Extrapolate (in-paint!?)





Cross-Correlation Expressed as a Sum

$$
J = CC(I,T) = \begin{bmatrix} J_0 \\ J_1 \\ \vdots \\ J_w \\ J_{w+1} \\ \vdots \\ \\ J_{M-1} \end{bmatrix}
=
\begin{bmatrix}
T_0 & T_1 & \cdots & T_w & 0 & \cdots & \cdots & 0 \\
T_{-1} & T_0 & T_1 & & T_w & & & 0 \\
\vdots & & & & & & & \\
T_{-w} & \cdots & T_0 & \cdots & T_w & 0 & \cdots & 0 \\
0 & T_{-w} & & T_0 & \cdots T_{w-1} & T_w & \cdots & 0 \\
\vdots & & & & & & & \\
 & 0 & 0 & T_{-w} & T_{-w-1} \cdots & T_{-1} & T_0 & T_1 \\
 & & 0 & \cdots & 0 & T_{-w} \cdots & T_{-2} \; T_{-1} & T_0
\end{bmatrix}
\begin{bmatrix} I_0 \\ I_1 \\ \vdots \\ I_w \\ I_{w+1} \\ \vdots \\ I_{2w} \\ \vdots \\ I_{M-1} \end{bmatrix}
$$

Cross-Correlation Expressed as a Sum

$$
J = cc(I,T) = \begin{bmatrix} J_0 \\ J_1 \\ \vdots \\ \hdashline J_w \\ J_{w+1} \\ \vdots \\ \hdashline \\ J_{M-1} \end{bmatrix} = \begin{bmatrix} T_0 & T_1 & \cdots & T_w & 0 & \cdots & \cdots & 0 \\ T_{-1} & T_0 & T_1 & & T_w & & & 0 \\ \hdashline T_{-w} & \cdot & T_0 & \cdots & T_w & 0 & \cdot & 0 \\ 0 & T_{-w} & & T_0 & T_{w-1} & T_w & \cdot & 0 \\ \\ \hdashline \\ & & 0 & 0 & T_{-w} & T_{-w+1} \cdots T_{-1} & T_0 & T_1 \\ & & & 0 \cdots 0 & T_{-w} & \cdots T_{-2} & T_{-1} & T_0 \end{bmatrix} \begin{bmatrix} I_0 \\ I_1 \\ \vdots \\ I_w \\ I_{w+1} \\ \vdots \\ I_{2w} \\ \vdots \\ I_{M-1} \end{bmatrix}
$$

General
sum notation:

$$J_i = \sum_{k=0}^{M-1} I_k T_{k-i}$$

$0 \leq i \leq M-1$

Cross-Correlation Expressed as a Sum

$$J = CC(I,T) = \begin{bmatrix} J_0 \\ J_1 \\ \vdots \\ J_w \\ J_{w+1} \\ \vdots \\ \\ J_{M-1} \end{bmatrix} = \begin{bmatrix} T_0 & T_1 & \cdots & T_w & 0 & \cdots & \cdots & 0 \\ T_{-1} & T_0 & T_1 & & T_w & & & 0 \\ T_{-w} & \cdot & T_0 & \cdots & T_w & 0 & \cdot & 0 \\ 0 & T_{-w} & & T_0 & T_{w-1} & T_w & \cdot & 0 \\ & & & & & & & \\ & & 0 & 0 & T_{-w} & T_{-w-1} \cdots T_{-1} & T_0 & T_1 \\ & & & 0 \cdots 0 & T_{-w} & \cdots T_{-2} & T_{-1} & T_0 \end{bmatrix} \begin{bmatrix} I_0 \\ I_1 \\ \vdots \\ I_w \\ I_{w+1} \\ \vdots \\ I_{2w} \\ \vdots \\ I_{M-1} \end{bmatrix}$$

General sum notation:

$$J_i = \sum_{k=0}^{M-1} I_k T_{k-i}$$

$0 \leq i \leq M-1$

for instance:

$$J_w = \sum_{k=0}^{M-1} I_k \cdot T_{k-w}$$

$$J_{w+1} = \sum_{k=0}^{M-1} I_k \cdot T_{k-w-1} = \sum_{k=0}^{M-1} I_k \cdot T_{k-(w+1)}$$

Indexing by image position, not template position

$$J = cc(I,T) = \begin{bmatrix} J_0 \\ J_1 \\ \vdots \\ J_w \\ J_{w+1} \\ \vdots \\ \\ J_{M-1} \end{bmatrix} = \begin{bmatrix} T_0 & T_1 & \cdots & T_w & 0 & \cdots & \cdots & 0 \\ T_{-1} & T_0 & T_1 & & T_w & & & 0 \\ T_{-w} & \cdot & T_0 & \cdots & T_w & 0 & \cdot & 0 \\ 0 & T_{-w} & & T_0 & T_{w-1} & T_w & \cdot & 0 \\ & & & & & & & \\ & & 0 & 0 & T_{-w} & T_{-w+1} \cdots T_{-1} & T_0 & T_1 \\ & & & 0 & \cdots 0 & T_{-w} \cdots T_{-2} & T_{-1} & T_0 \end{bmatrix} \begin{bmatrix} I_0 \\ I_1 \\ \vdots \\ I_w \\ I_{w+1} \\ \vdots \\ I_{2w} \\ \vdots \\ I_{M-1} \end{bmatrix}$$

General sum notation:

$$J_i = \sum_{k=0}^{M-1} I_k T_{k-i}$$

$0 \leq i \leq M-1$

Equivalent to:

$$\boxed{J_i = \sum_{\ell=0}^{M-1} I_{\ell+i} T_\ell \qquad 0 \leq i \leq M-1}$$

* obtained by substituting $\ell = k - i$ in General Sum Notation formula

The Convolution Operation

A similar operation to Cross Correlation is Convolution:

Cross-correlation:

$$J_i = \sum_{k=0}^{M-1} I_k \cdot T_{k-i}$$

Convolution:

$$(I * T)_i = \sum_{k=0}^{M-1} I_k \cdot T_{i-k}$$

$$\begin{bmatrix} T_{-w} & T_{-w+1} & \cdots & T_0 & \cdots & T_w & 0 & \cdots & 0 \end{bmatrix}$$

$$\begin{bmatrix} T_{w} & T_{w-1} & \cdots & T_0 & \cdots & T_{-w} & 0 & \cdots & 0 \end{bmatrix}$$

$$\begin{bmatrix} I_0 \\ I_1 \\ I_2 \\ \vdots \\ I_w \\ \vdots \\ I_{2w} \\ I_{2w+1} \\ \vdots \\ I_{M-1} \end{bmatrix}$$

The Convolution Operation

The convolution operation is one of the most fundamental operations in image (and signal) processing.

In this context

I is the “image” or the “signal”

T is the “filter”, “mask”, “template”, “impulse response”, “kernel”…

Notation: $I * T$: reads as “the convolution of I with T” an means:

Convolution:

$$(I * T)_i = \sum_{k=0}^{M-1} I_k \cdot T_{i-k}$$

The Convolution Properties

1. For symmetric masks, convolution is equal to cross-correlation:

$$\boxed{\begin{array}{l} CC(I,T) = I * T \\ \text{when } \; T_i = T_{-i} \end{array}}$$

2. Commutativity:

$$\boxed{I * T = T * I}$$

3. Linearity:

$$\boxed{\begin{array}{l} (aI + bJ) * T = \\ \qquad a(I * T) + b(J * T) \\ \text{for any constants } a, b \end{array}}$$

You may want to prove these as an exercise.

The Convolution Properties

But images are 2D!

The Convolution Properties

Similar to 2D cross-correlation. For a M by N image:

$$(I * T)(i,j) = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} I(k,l) \cdot T(i-k, j-l)$$

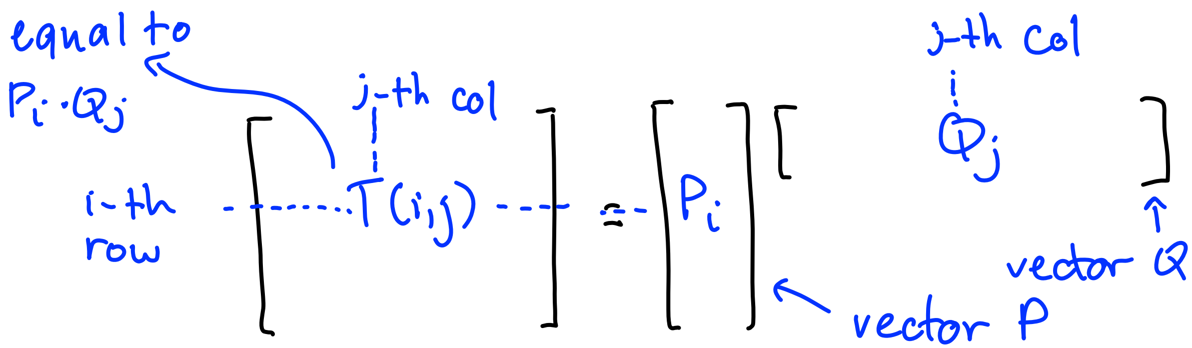
This can get expensive. If the image is M by N and the Template is P by Q, the complexity is:

O(MNPQ)

The Separable Convolution

There is one very special case, when the Template is "separable".

Separable templates are such that $T=PQ^T$ for some vectors P, Q.



The Separable Convolution

There is one very special case, when the Template is "separable".

For instance, is the "Sobel" kernel separable?

$$\text{Sobel} = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} = \begin{bmatrix} \\ \\ \\ \end{bmatrix} \begin{bmatrix} & & \end{bmatrix}$$

The Separable Convolution

There is one very special case, when the Template is "separable".

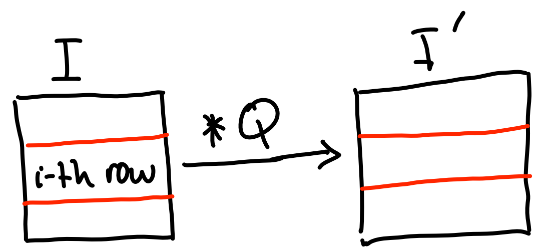
For instance, is the "Sobel" kernel separable?

$$\text{Sobel} = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} \begin{pmatrix} [-1 & 0 & 1] \end{pmatrix}$$

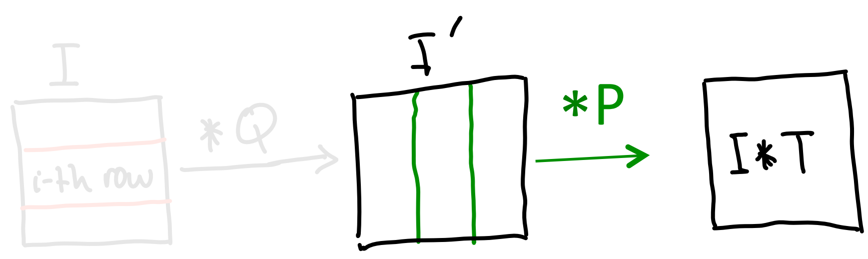
The Convolution Operation

When a kernel is separable the 2D convolution can be obtained from 2, cascaded 1D convolutions. The algorithm is as follows:

1. Compute the 1D convolution between each row of I and Q to obtain I'.



2. Compute the 1D convolution between each row of I' and P, to obtain the final 2D result:



The Convolution Operation

When a kernel is separable the 2D convolution can be obtained from 2, cascaded 1D convolutions.

In the previous slide, we first convolved rows of I with Q to obtain I’ and then columns of I’ with P to obtain the final result.

The same result can be obtained by first convolving columns of I with P to obtain I’’ and then convolve rows of I’’ with Q.

You may want to prove this as an exercise

Topic 6.1:

Gaussian & Laplacian Pyramids

- The gaussian pyramid (intro)
- The convolution operation
- Constructing the gaussian pyramid
  - The REDUCE() function
- Constructing the Laplacian pyramid
  - The EXPAND() function

The Gaussian Pyramid

The Gaussian Pyramid: A representation in multiple scales

Original Image

The Gaussian Pyramid

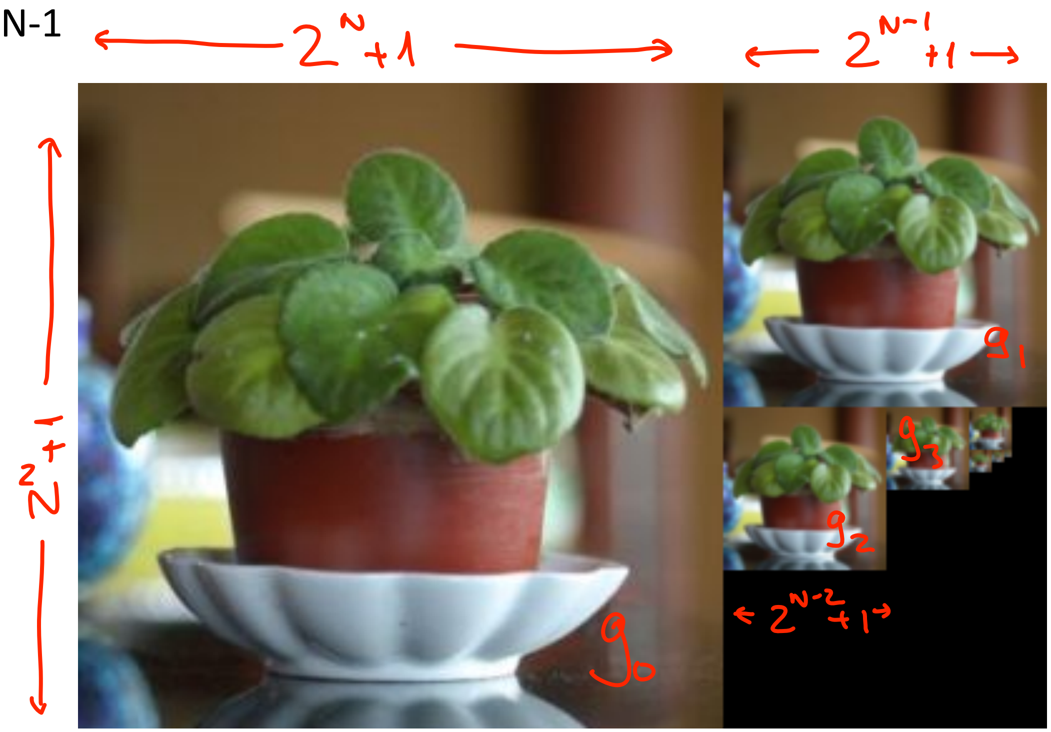
The elements of a Gaussian Pyramids are smoothed copies of the image at different scales.

Input: Image I of size $(2^N+1)x(2^N+1)$

Output: Images $g_0, g_1, \ldots, g_{N-1}$

where the size of g_i is:

$$(2^{N-i}+1)x(2^{N-i}+1)$$



Building (or computing) a Gaussian Pyramid

To generate a Gaussian pyramid,
iterate between these two steps:

Smoothing: Remove high-frequency components that could cause aliasing.

Down-sampling: Reduce the image size by ½ at each level.

Operation #1: Smooth Image, recursively

Take the original image g_0 and compute g_1 using:

g_0

$$\boxed{\hat{g}_1(i,j) = \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m,n) \cdot g_0(i-m, j-n)}$$

$$\hat{g}_1 = w * g_0$$

a 5x5 filter like

1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25

Operation #1: Smooth Image at N-1 Scales

To estimate g_2 repeat using g_1 as the input:

$\hat{g}_1$

$$\hat{g}_2 = w * \hat{g}_1$$

$$= w * (w * g_0)$$

$$= \underbrace{(w * w)}_{} * g_0$$

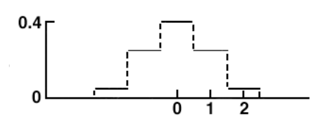
can be thought of as a filter

$h = w * w$

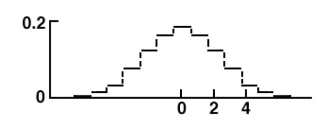
whose radius is twice that of w

In 1D:

w:



w*w:



Operation #1: Smooth Image at N-1 Scales

To estimate g_3 repeat using g_2 as the input:

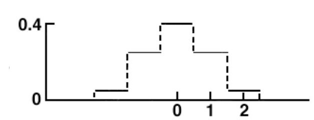
$$\hat{g}_3 = w * \hat{g}_2$$

$$= \underbrace{(w * w * w)}_{\text{radius is 4 times that of } w} * g_0$$

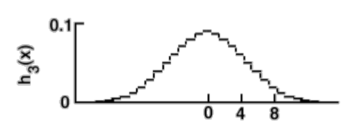
$\hat{g}_2$

In 1D:

w:



w*w*w:



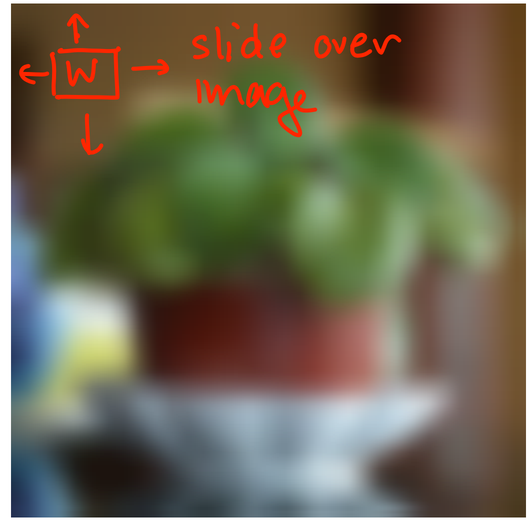
Operation #1: Smooth Image at N-1 Scales

And so on…

$$\hat{g}_4 = w * \hat{g}_3$$

$$= (w * w * w * w) * g_0$$

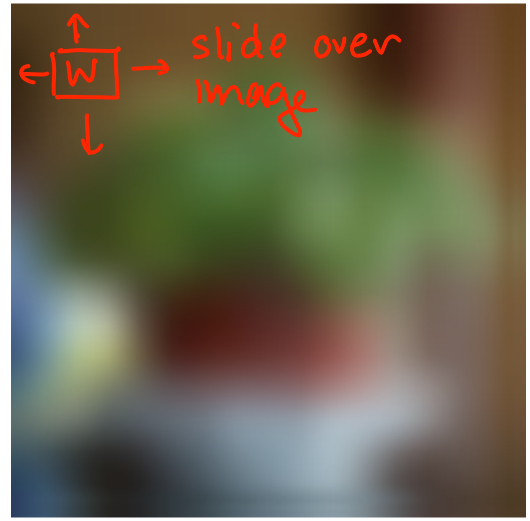
$\hat{g}_3$



Operation #1: Smooth Image at N-1 Scales

And so on…

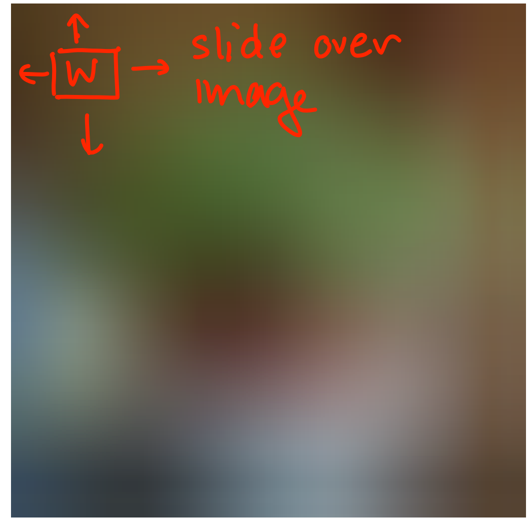
$\hat{g}_4$



Operation #1: Smooth Image at N-1 Scales

And so on…

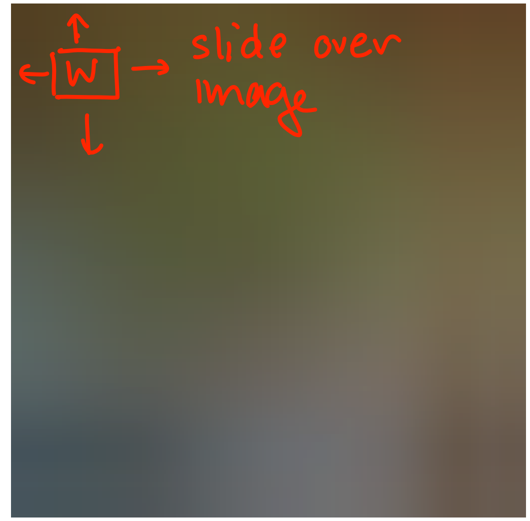
$\hat{g}_5$



Operation #1: Smooth Image at N-1 Scales

And so on…

$\hat{g}_6$



Operation #1: Smooth Image at N-1 Scales

$\hat{g}_7$

2D Kernel to 1D kernel, how?

To take a 5x5 2D kernel into a 1D kernel we must satisfy 4 criteria:

1. The window size must remain the same: 5 elements.
2. The kernel must be symmetric around its origin

$$\hat{w} = \begin{bmatrix} c \\ b \\ a \\ b \\ c \end{bmatrix}$$

3. Applying w to a constant image does not change (intensity preserving)

$$\sum_{m=-2}^{2} \hat{w}(m) = 1 \iff a + 2b + 2c = 1$$

4. The ratio of the contributions should follow:

$$a + 2c = 2b = 1/2$$

2D Kernel to 1D kernel, how?

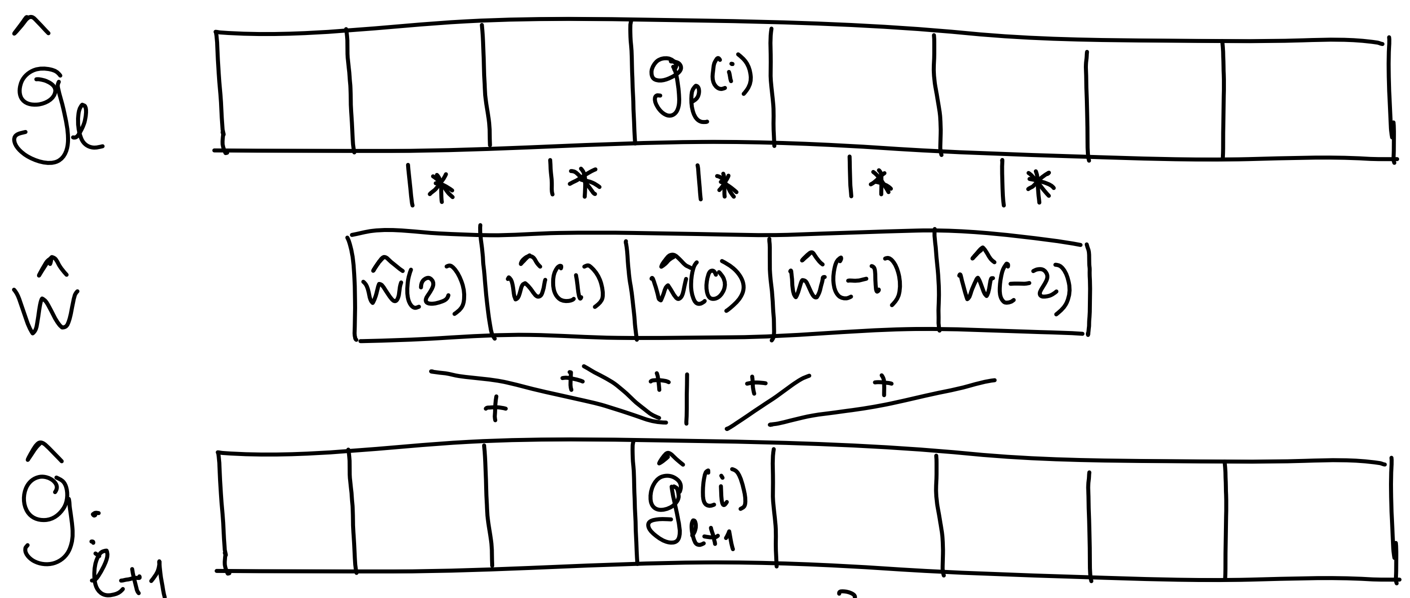
The conditions on the previous slide render filters of the form:

$$\hat{w} = \begin{bmatrix} c \\ b \\ a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1/4 - a/2 \\ 1/4 \\ a \\ 1/4 \\ 1/4 - a/2 \end{bmatrix}$$

usually $a \in [0.3, 0.6]$

2D Kernel to 1D kernel, how?

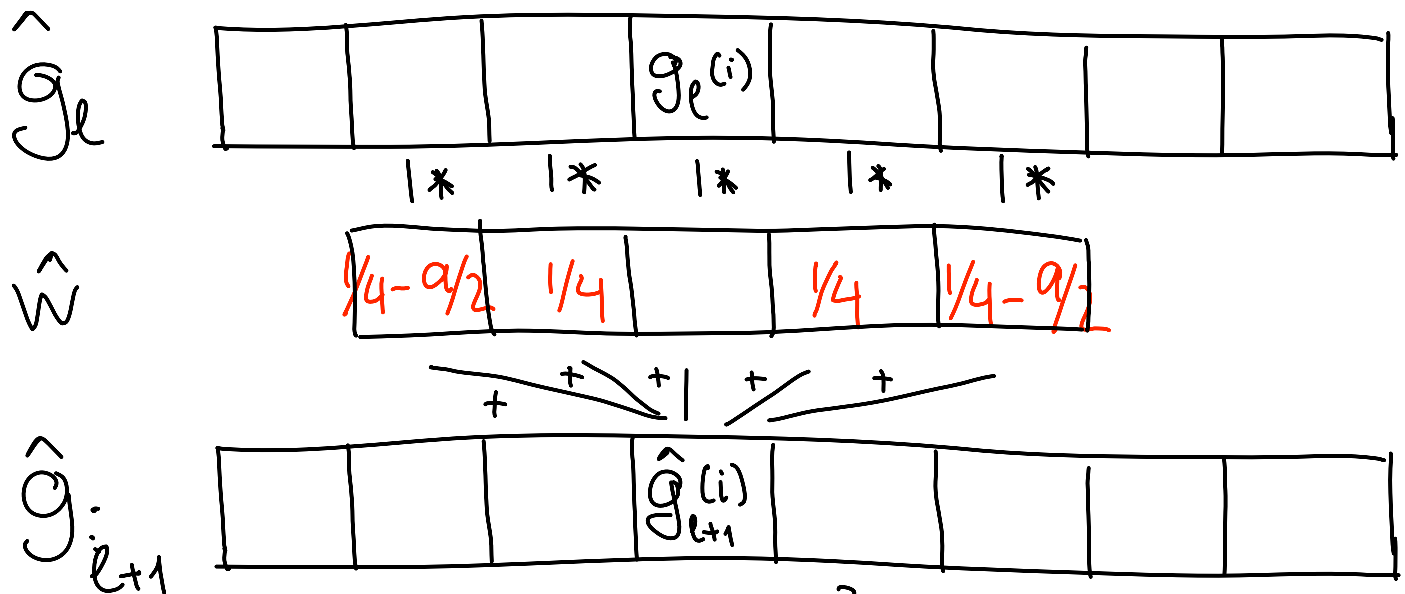
This means that to estimate the i^{th} pixel at the $(l+1)^{th}$ level one must do:



$$\hat{g}_{l+1}(i) = \sum_{m=-2}^{2} \hat{w}(m) \cdot \hat{g}_l(i-m)$$

2D Kernel to 1D kernel, how?

This means that to estimate the i^{th} pixel at the $(l+1)^{th}$ level one must do:

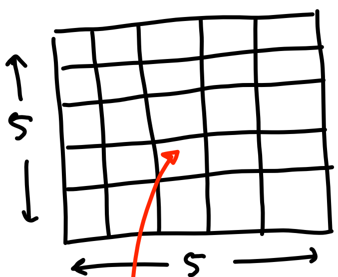


$$\hat{g}_{l+1}(i) = \sum_{m=-2}^{2} \hat{w}(m) \cdot \hat{g}_l(i-m)$$

Defining the Smoothing Filter in 2D

$$\hat{g}_1 = w * g_0$$

w is a separable smoothing filter defined by $\hat{w}$



$$= \begin{bmatrix} \hat{w} \end{bmatrix} \begin{bmatrix} (\hat{w})^T \end{bmatrix}$$

5x1 vector

$$w(m,n) = \hat{w}(m) \cdot \hat{w}(n)$$

g_0

Exploiting separability to compute $\hat{g}_1$:

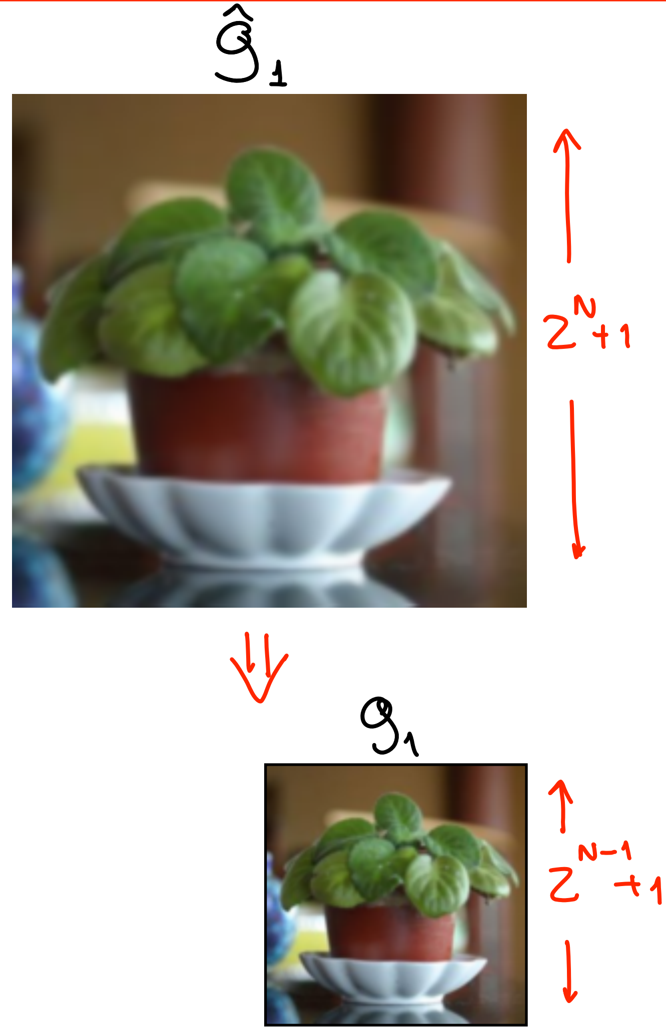
1. Convolve each row of g_0 with $\hat{w}$
2. Convolve the columns of the result with $\hat{w}$ again

Operation #2: Downsample the Smoothed Image

Downsample by taking every other pixel.

Smoothing prevents aliasing effects.

$$\boxed{g_1(i,j) = \hat{g}_1(2i,2j)}$$



Topic 6.1:

Gaussian & Laplacian Pyramids

- The gaussian pyramid (intro)
- The convolution operation
- Constructing the gaussian pyramid
  - The REDUCE() function
- Constructing the Laplacian pyramid
  - The EXPAND() function

Operations #1 & #2: The REDUCE() Function

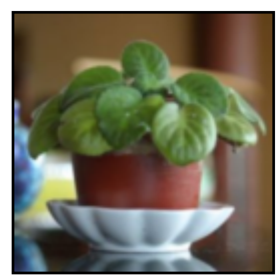
The REDUCE() function combines smoothing and down sampling.

g_ℓ

$\longleftarrow 2^{N-\ell}+1 \longrightarrow$

$w * g_\ell \Longrightarrow$

$\Downarrow$ down sample x2



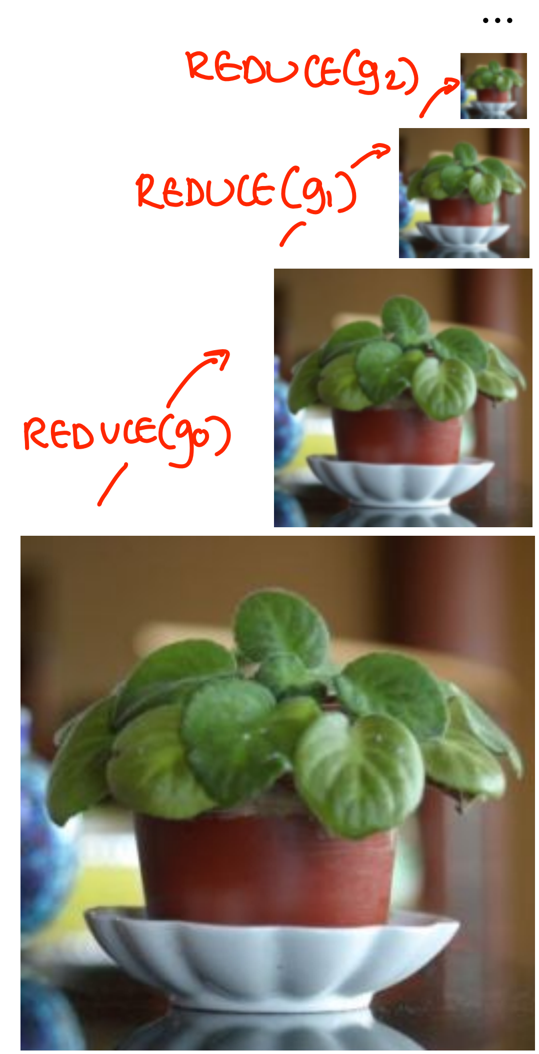
$g_{\ell+1}$

$\leftarrow 2^{N-\ell-1}+1 \rightarrow$

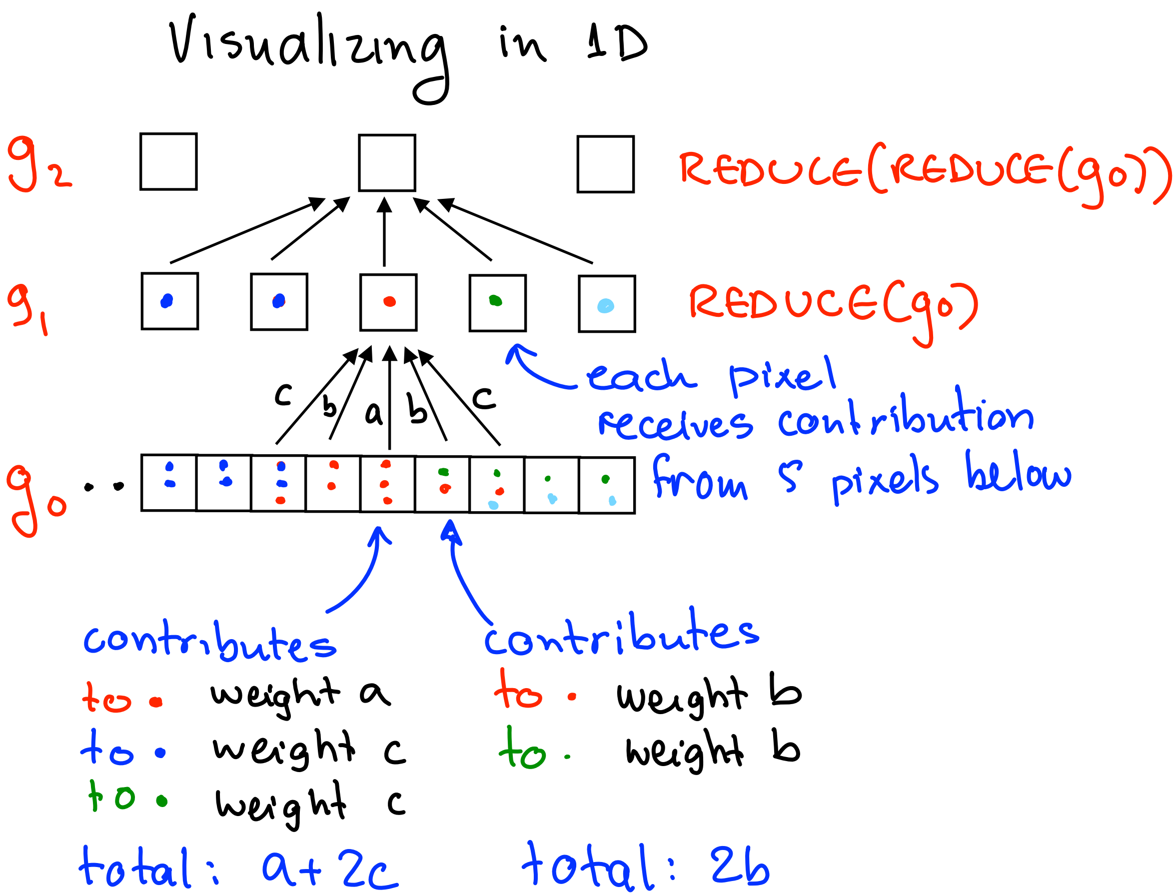
$$g_{\ell+1} = \mathrm{REDUCE}(g_\ell)$$

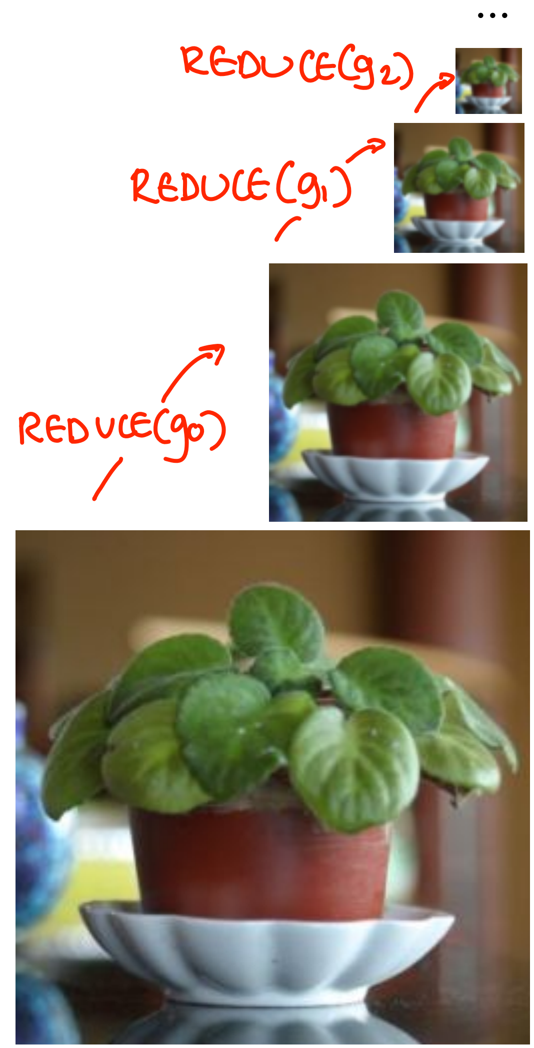
$$\boxed{g_{\ell+1}(i,j) = \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m,n) \cdot g_\ell(2i-m, 2j-n)}$$

The REDUCE() function

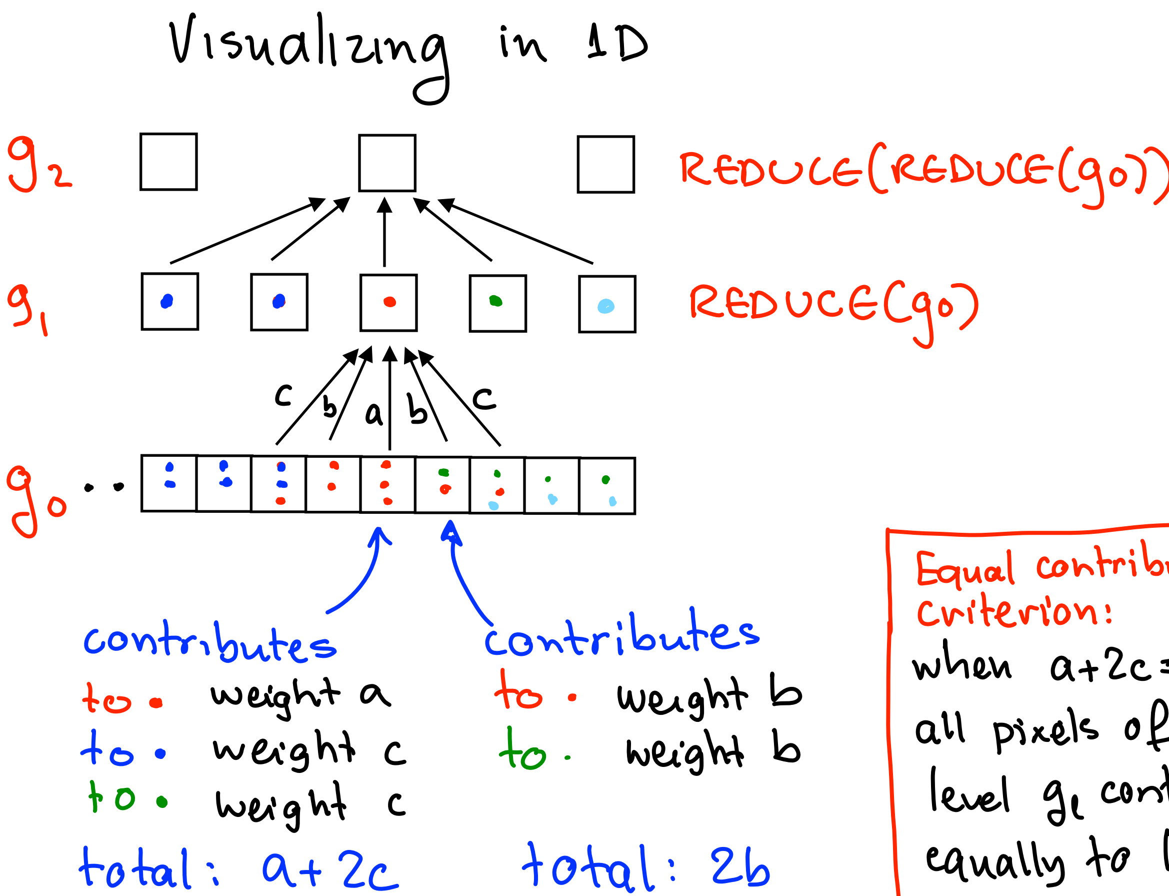


The REDUCE() function

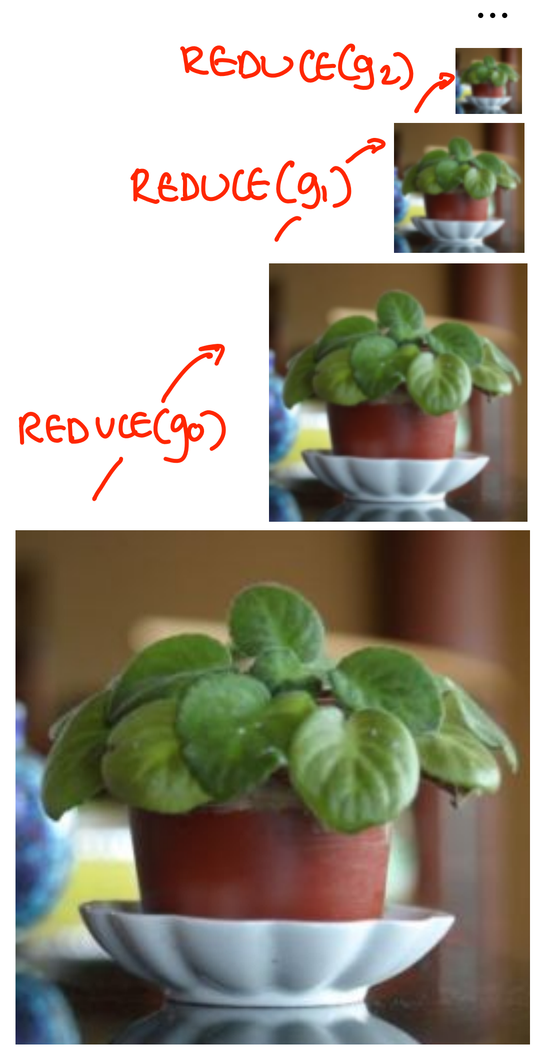




The REDUCE() function

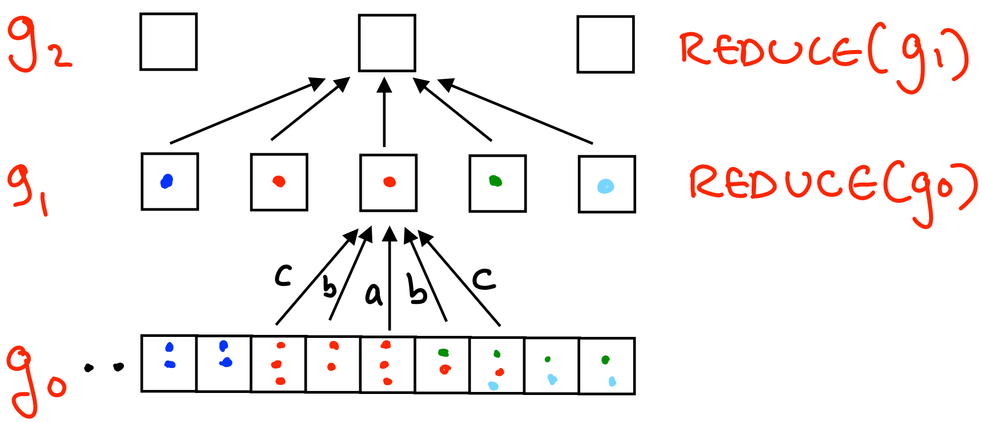


Equal contribution criterion:
when $a+2c=2b$
all pixels of level g_l contribute equally to level g_{l+1}



The REDUCE() function

Visualizing in 1D



REDUCE function in matrix notation (1D)

$$\overset{2^{N-1}+1}{\begin{bmatrix} g_1 \end{bmatrix}} = \overset{\leftarrow 2^N+1 \rightarrow}{\begin{bmatrix} 1 & 0 & 0 & \cdots & & & 0 \\ 0 & 0 & 1 & \cdots & & \cdots & 0 \\ 0 & 0 & 0 & 0 & 1 & \cdots & 0 \\ & & \cdots & & & & \end{bmatrix}} \overset{\leftarrow 2^N+1 \rightarrow}{\begin{bmatrix} c & b & a & b & c & & 0 \\ & c & b & a & b & c & \\ & & & \ddots & \ddots & & \\ 0 & & & & \ddots & \ddots & \end{bmatrix}} \begin{bmatrix} g_0 \end{bmatrix}$$

downsampling matrix D_0

convolution matrix C_0

$$\boxed{g_1 = D_0 \cdot C_0 g_0}$$

General expression:

$$g_{\ell+1} = D_\ell \cdot C_\ell \cdot g_\ell$$

What Does Smoothing Take Away?

$g_0 = I$

original photo

What Does Smoothing Take Away?

smoothed photo

$\hat{g}_1 = w * g_0$

What Does Smoothing Take Away?

$g_0 - \hat{g}_1$

Details in g_0 that were <u>not</u> represented in $\hat{g}_1$

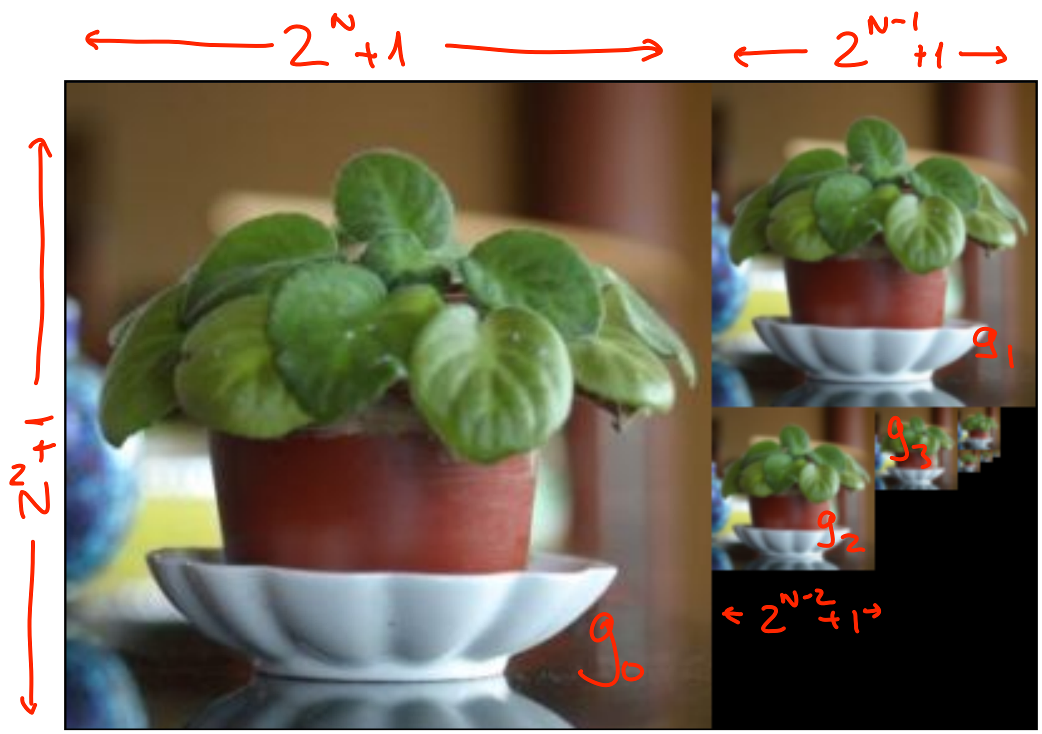
Topic 6.1:

Gaussian & Laplacian Pyramids

- The gaussian pyramid (intro)
- The convolution operation
- Constructing the gaussian pyramid
  - The REDUCE() function
- Constructing the Laplacian pyramid
  - The EXPAND() function

The Laplacian Pyramid

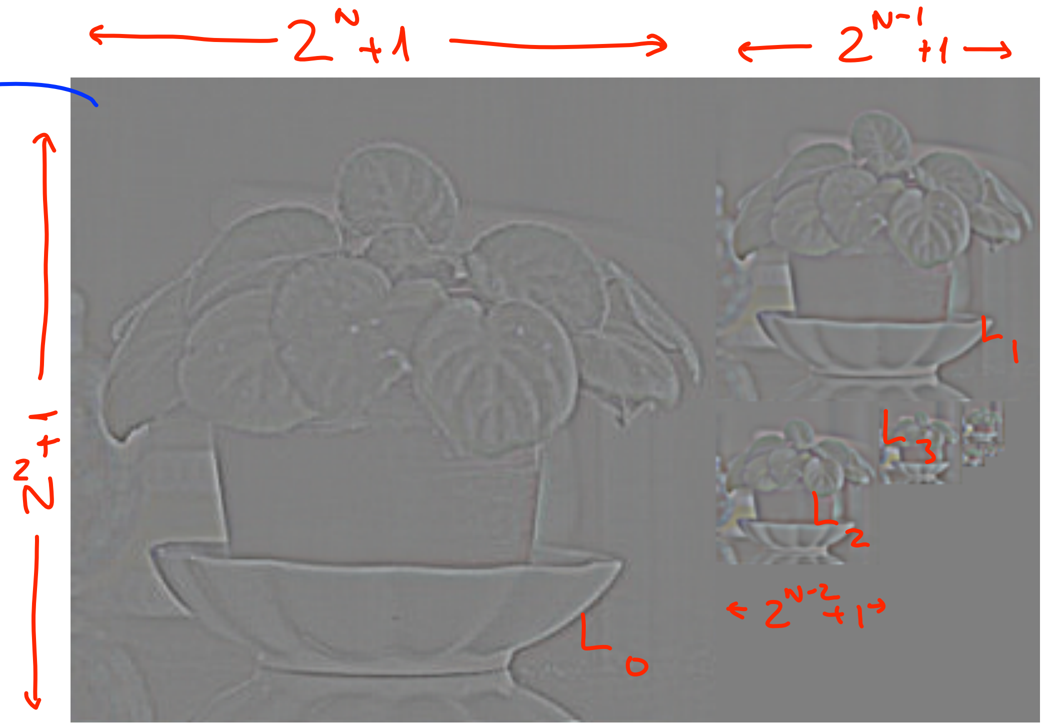
What if instead of storing the smoothed images, we store only the difference between the levels g_l and g_{l+1}



The Laplacian Pyramid

What if instead of storing the smoothed images, we store only the difference between the levels g_l and g_{l+1}

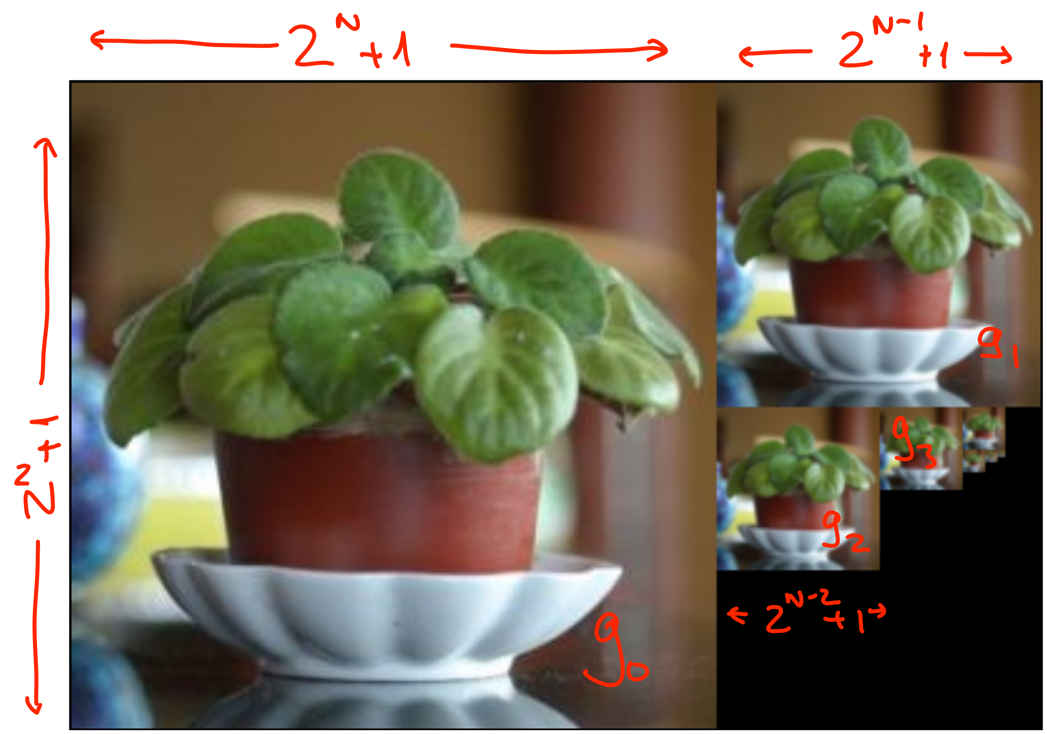
most values are 0 (gray) ⇒ can be quantized & represented with few bits



The Laplacian Pyramid

What if instead of storing the smoothed images, we store only the difference between the levels g_l and g_{l+1}

- But $g_{\ell+1}, g_\ell$ are not the same size!
  $\Rightarrow$ Expand $g_{\ell+1}$ to make it equal size to g_ℓ



Operation #3: The EXPAND() Function

And we can write this with an equation:

$\leftarrow 2^{N-\ell}+1 \rightarrow$

g_ℓ

EXPAND( )

$\Longrightarrow$

$\leftarrow 2^{N-\ell+1}+1 \rightarrow$

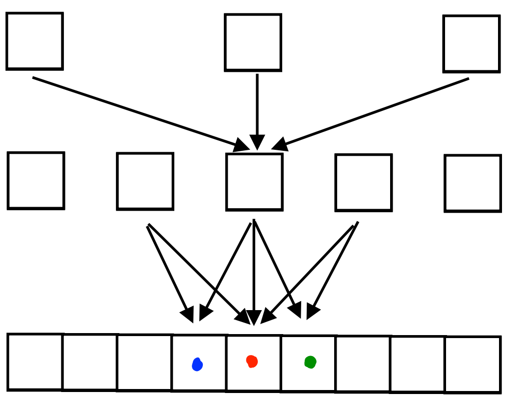
EXPAND(g_ℓ)

$$\boxed{\mathrm{EXPAND}(g_\ell) = 4\sum_{m=-2}^{2}\sum_{n=-2}^{2} w(m,n)\, g_\ell\left(\frac{i-m}{2}, \frac{j-n}{2}\right)}$$

The EXPAND() function

- The function upsamples level g_ℓ by doubling its size from $(2^{N-\ell}+1)\times(2^{N-\ell}+1)$ to $(2^{N-\ell+1}+1)\times(2^{N-\ell+1}+1)$

Visualizing in 1D



g_1

$\text{EXPAND}(g_1)$

EVEN pixels receive contribution from 3 pixels above them

(total weight = $2c+a=1/2$)

ODD pixels receive contribution from 2 pixels above them

(total weight = $2b=1/2$)

General expression in 1D

$$\text{EXPAND}(g_\ell)(i) = 2\sum_{m=-2}^{2}\hat{w}(m)\cdot g_\ell\left(\underbrace{\frac{i-m}{2}}\right)$$

evaluated only for i, m where $\frac{i-m}{2}$ is an integer

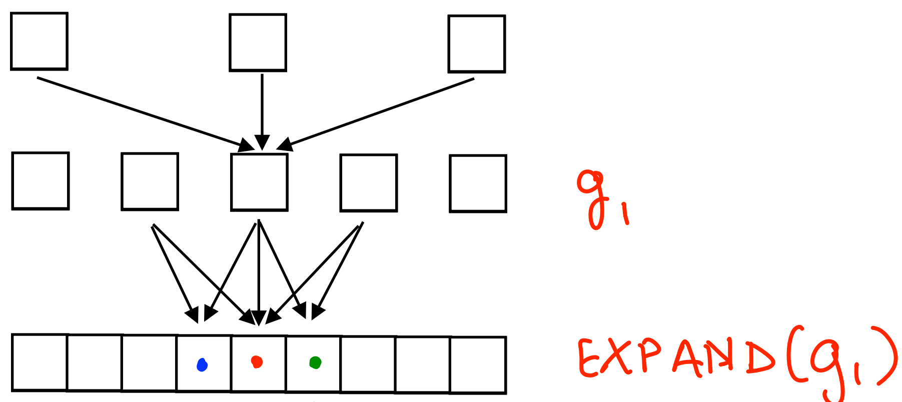
The EXPAND() function

- The function upsamples level g_ℓ by doubling its size from $(2^{N-\ell}+1)\times(2^{N-\ell}+1)$ to $(2^{N-\ell+1}+1)\times(2^{N-\ell+1}+1)$

The EXPAND() function

- The function upsamples level g_ℓ by doubling its size from $(2^{N-\ell}+1)\times(2^{N-\ell}+1)$ to $(2^{N-\ell+1}+1)\times(2^{N-\ell+1}+1)$

Visualizing in 1D



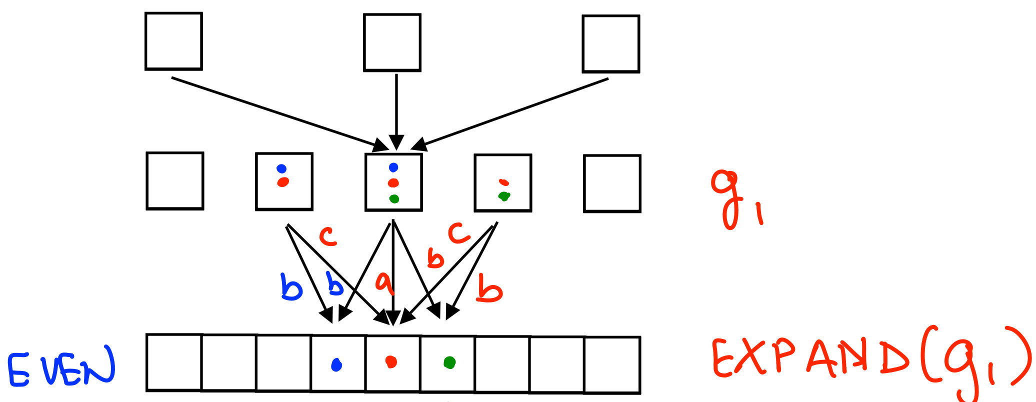
EVEN pixels receive contribution from 3 pixels above them (total weight $= 2c+a = 1/2$)

ODD pixels receive contribution from 2 pixels above them (total weight $= 2b = 1/2$)

The EXPAND() function

- The function upsamples level g_ℓ by doubling its size from $(2^{N-\ell}+1)\times(2^{N-\ell}+1)$ to $(2^{N-\ell+1}+1)\times(2^{N-\ell+1}+1)$

Visualizing in 1D



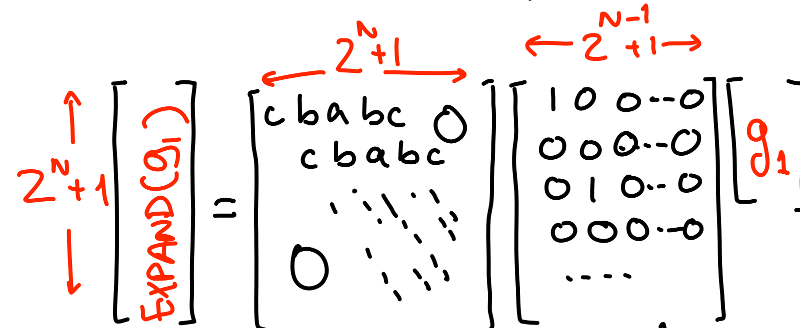
EVEN pixels receive contribution from 3 pixels above them

(total weight = $2c+a=1/2$)

ODD pixels receive contribution from 2 pixels above them

(total weight = $2b=1/2$)

EXPAND function in matrix notation (1D)



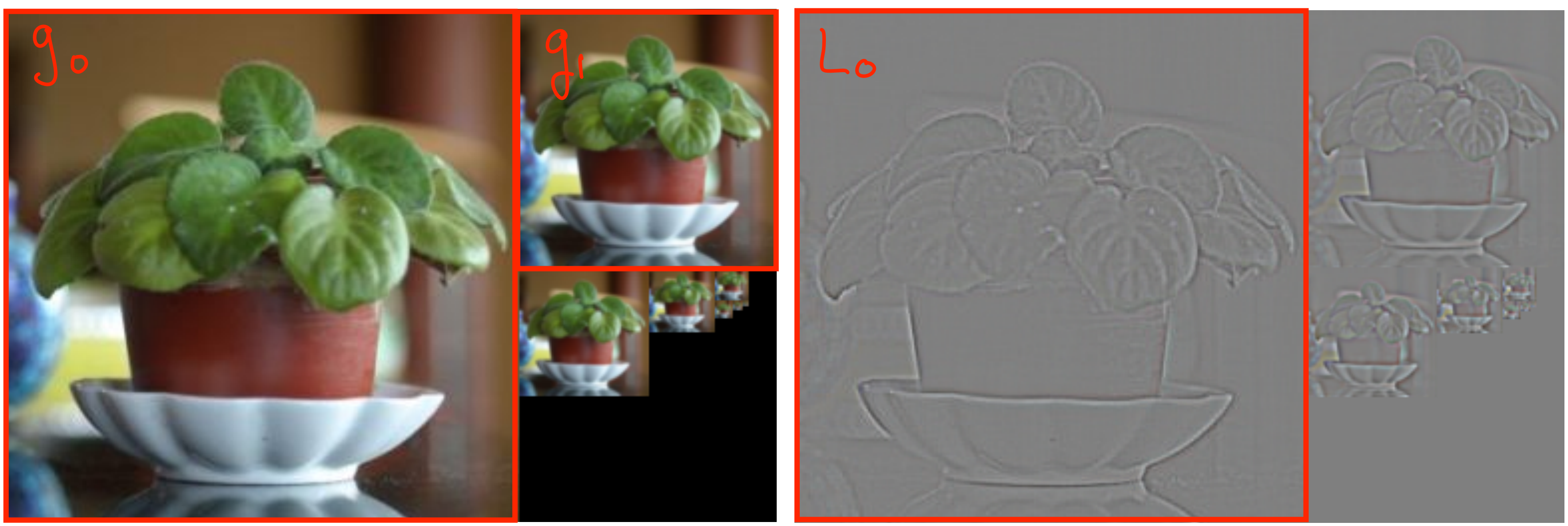
Convolution matrix C_0

upsampling matrix $(D_0)^T$

$$\boxed{\text{EXPAND}(g_1) = C_0 \cdot (D_0)^T g_1}$$

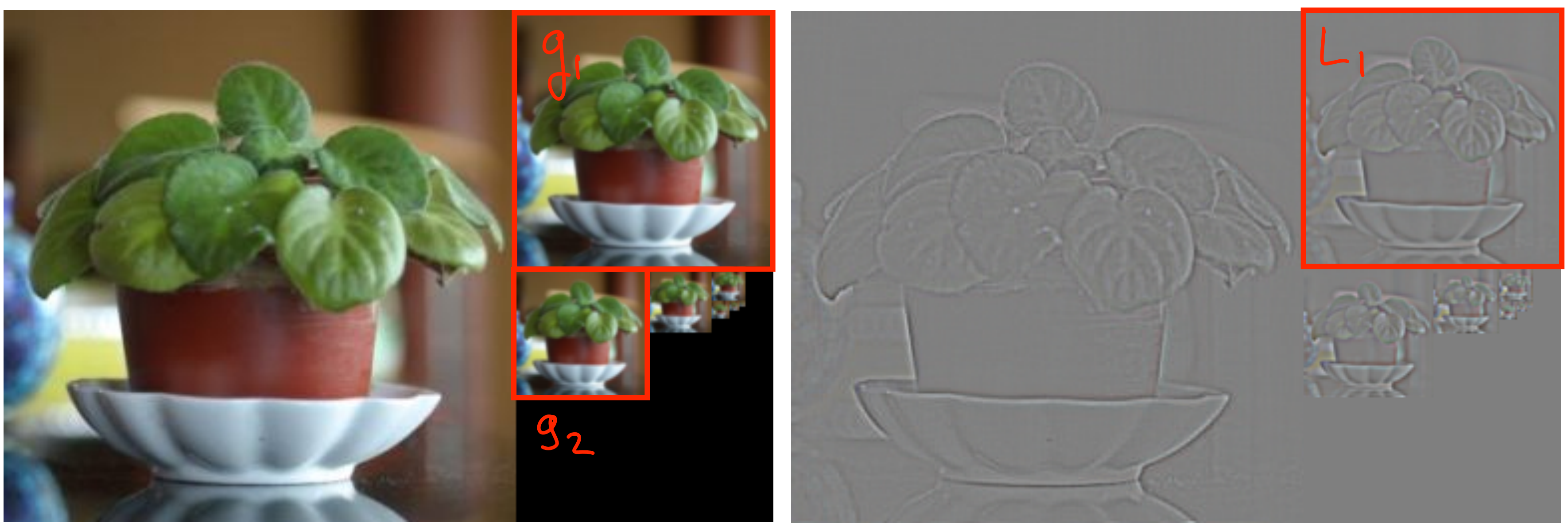
The Laplacian Pyramid

$$L_0 = g_0 - \text{EXPAND}(g_1)$$



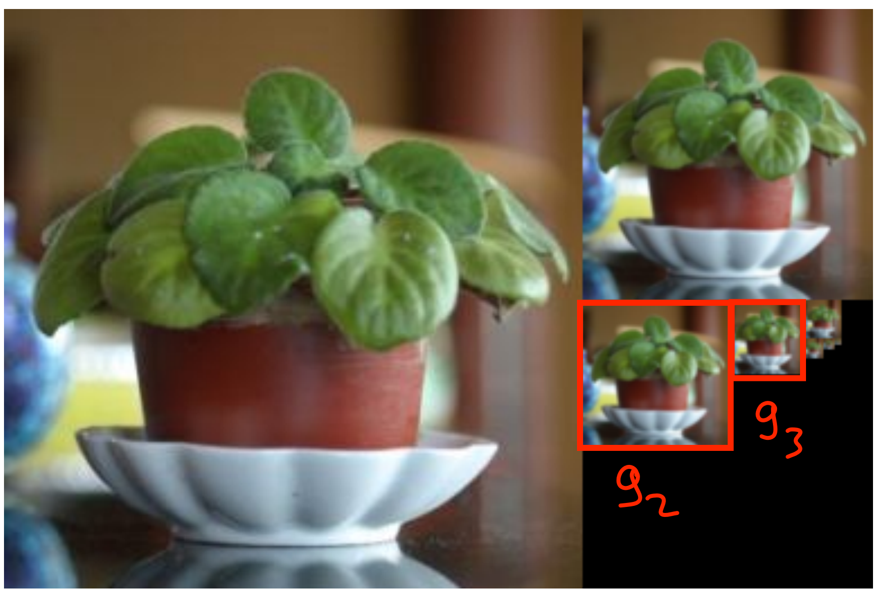
The Laplacian Pyramid

$L_1 = g_1 - \mathrm{EXPAND}(g_2)$



The Laplacian Pyramid

$L_2 = g_2 - \text{EXPAND}(g_3)$



The Laplacian Pyramid

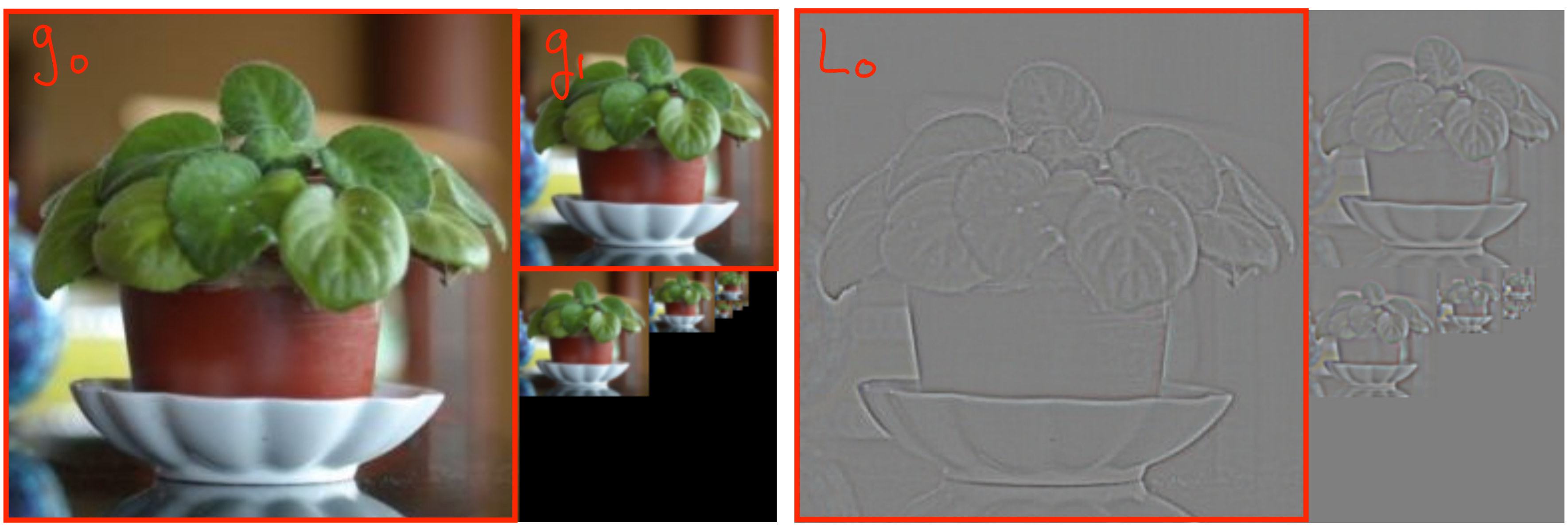
- Often we use a truncated Laplacian pyramid by storing images $L_0, L_1, \dots, L_k, g_{k+1}$ for $k+1 < N$

Base case: store g_N

The Laplacian Pyramid

But we can use the Pyramid in the opposite way too. To recover g_0, we can do:

$$g_0 = L_0 + \mathrm{EXPAND}\,(g_1)$$

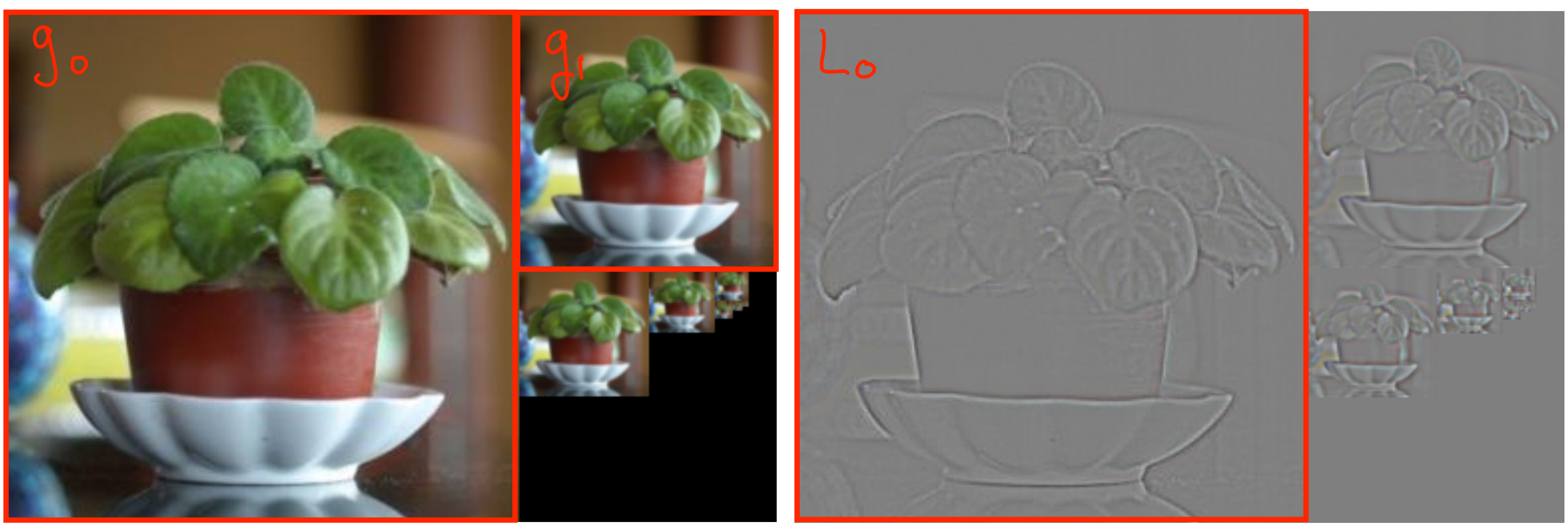


The Laplacian Pyramid

But we can use the Pyramid in the opposite way too. To recover g_0, we can do: $g_0 = L_0 + \text{EXPAND}(g_1)$

Given an input image of size $(2^N+1) \times (2^N+1)$, the Laplacian pyramid representation consists of

- $L_0, \ldots, L_{K-1}$
- g_K for some $K \leq N$



Transmission using EXPAND

RECEIVER

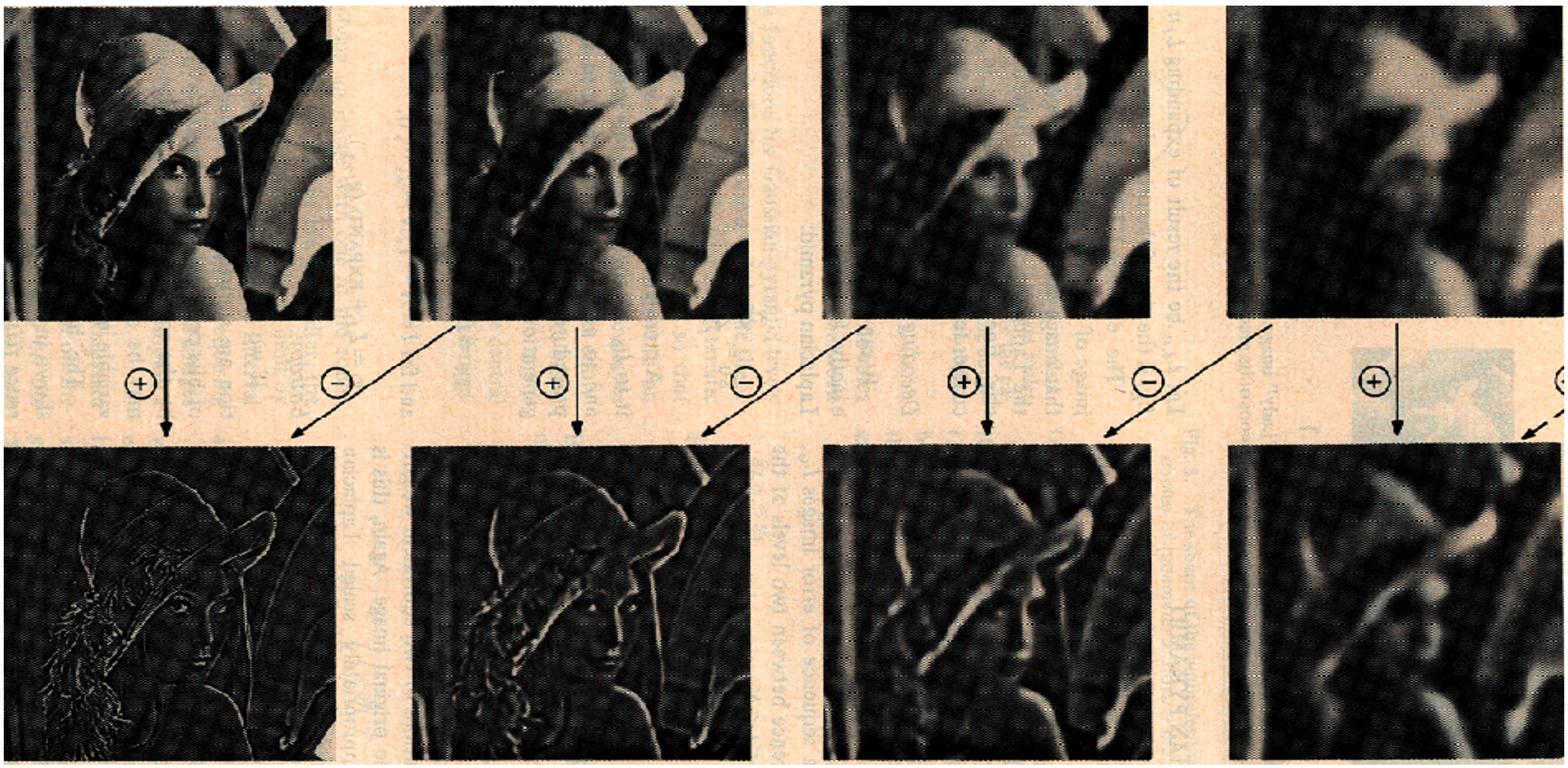
TRANSMITTER

③ Display $d_3 = d_2 + \text{EXPAND}^{k-2}(L_{k-2})$

② Display $d_2 = d_1 + \text{EXPAND}^{k-1}(L_{k-1})$

① Display $d_1 = \text{EXPAND}^{k}(g_k)$

① Transmit g_k



⑤ Transmit L_{k-4}

④ Transmit L_{k-3}

③ Transmit L_{k-2}

② Transmit L_{k-1}

Topic 6:

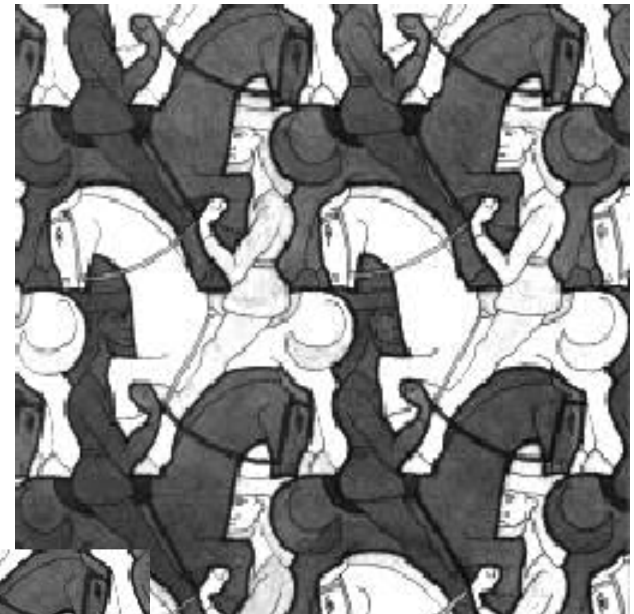
Hierarchical image representations

1. Gaussian & Laplacian pyramids
2. Applications:
    1. Multi-resolution image blending
    2. Multi-resolution image editing
    3. Multi-resolution texture synthesis

Image Blending

Source 1

Source 2

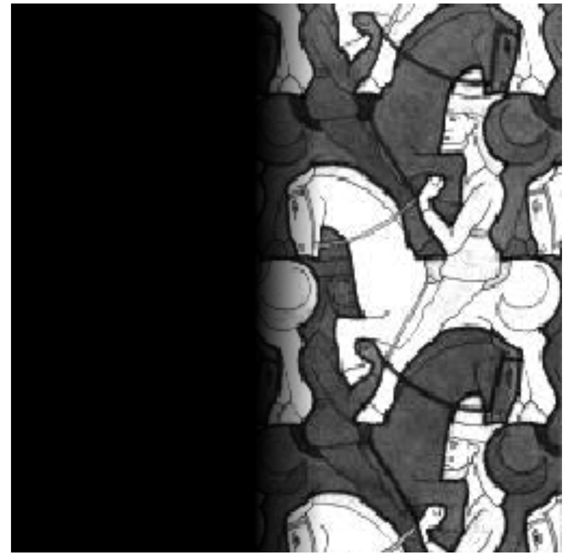


Blend

Slides adapted from A. Efros (CMU)

Feathering

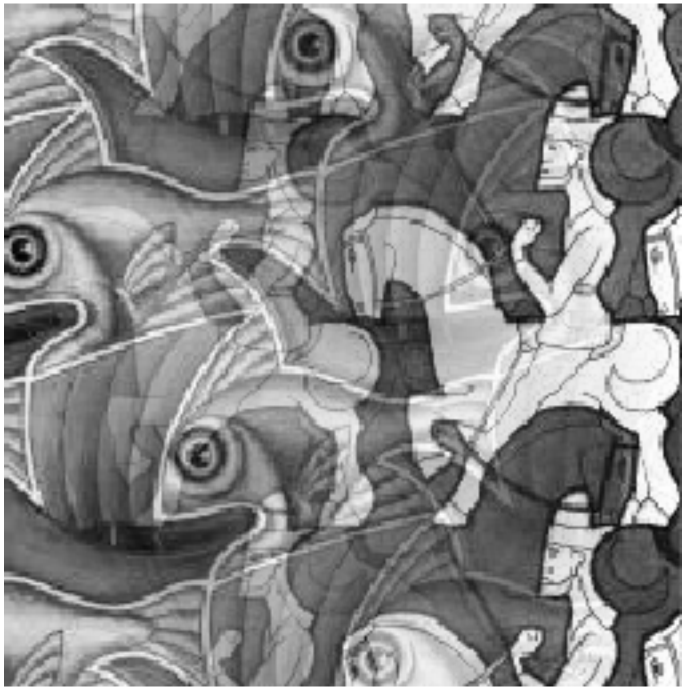
+

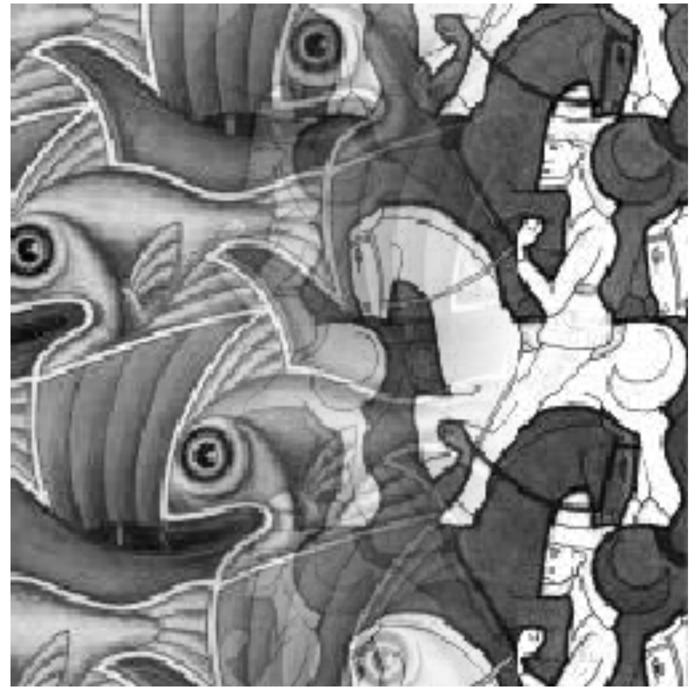


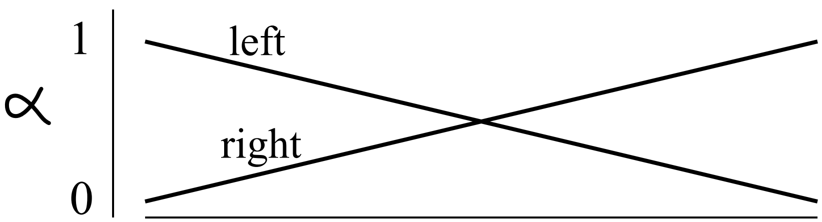
=

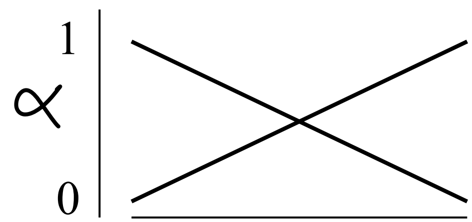
Assign an alpha value to each pixel near the seam to make it less visible

Effect of Window Size



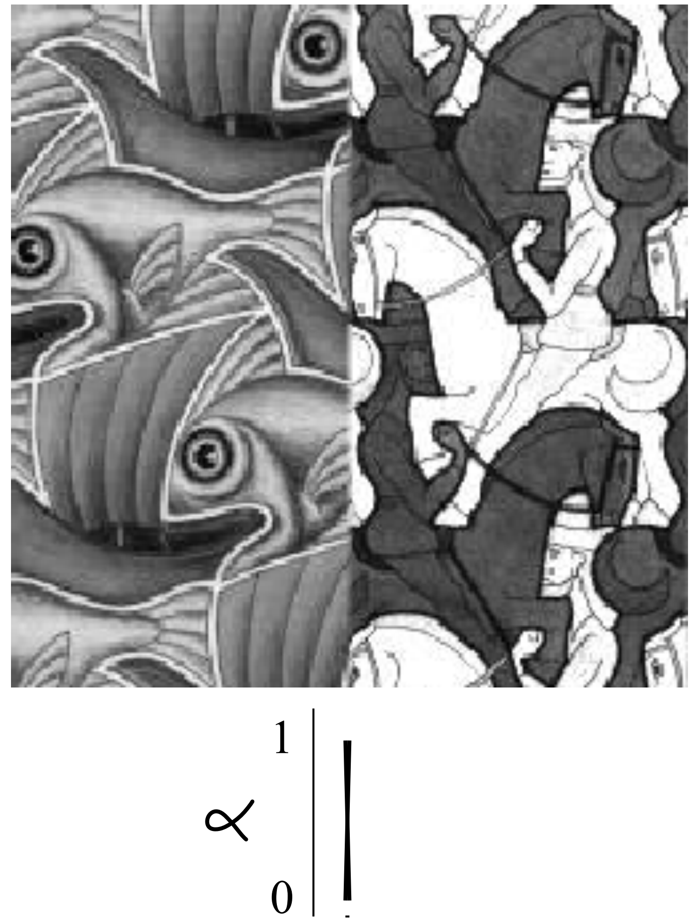






Ghosting is most apparent here

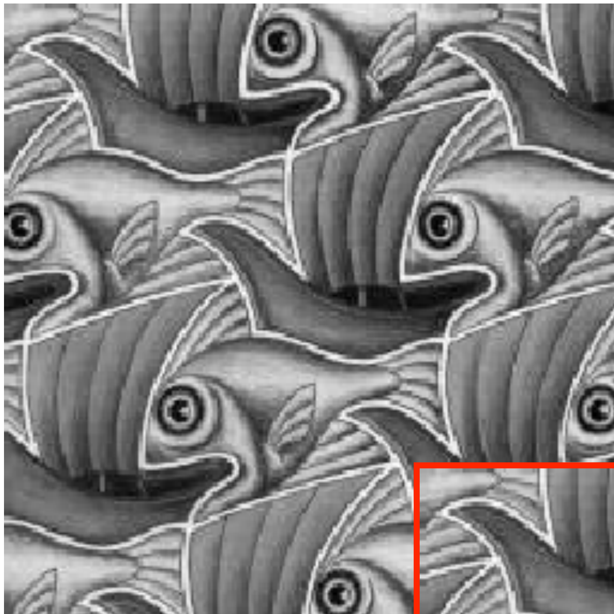
Effect of Window Size



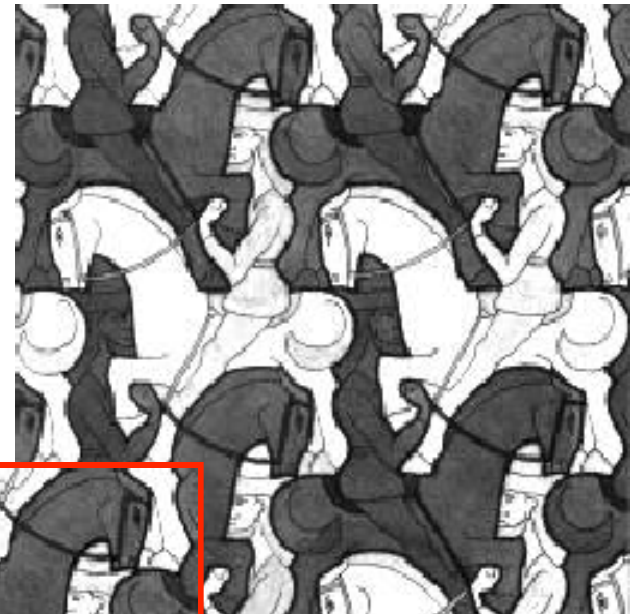
seam most visible
here

Image Blending

Source A



Source B



Blend

Main challenge:

- Minimize ghosting without making visible seams

Slides adapted from A. Efros (CMU)

Application #1: Pyramid Blending Algorithm

Input: Source images A, B & binary matte M.
Output: Blended image S.

A

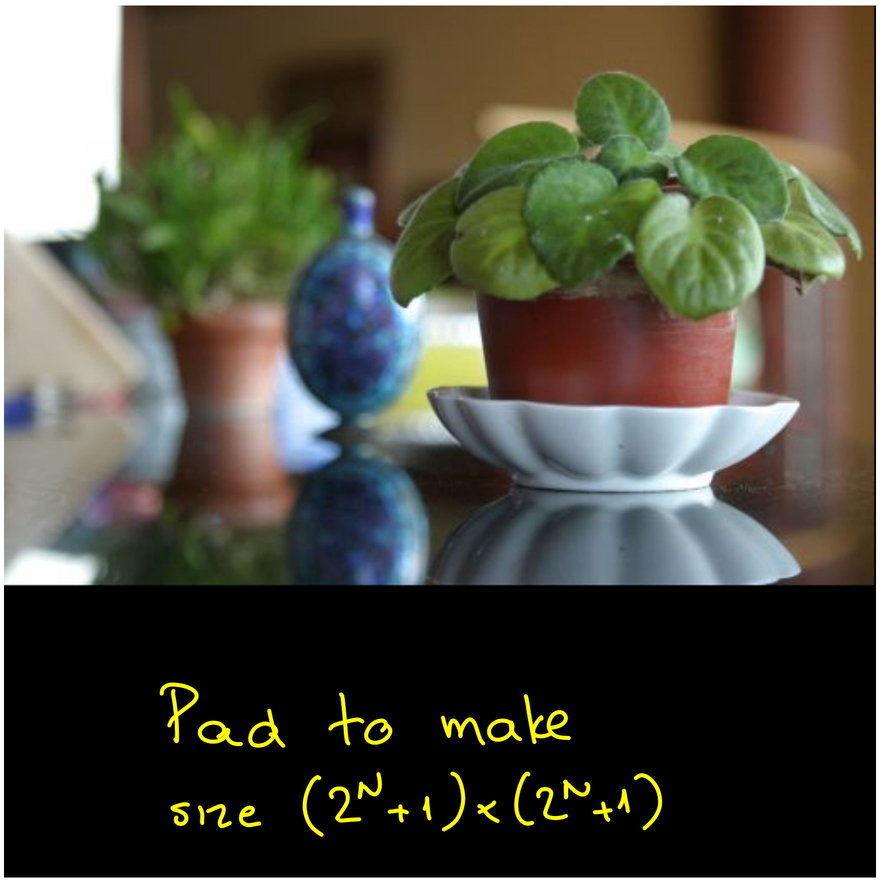
B

Pyramid Blending Algorithm

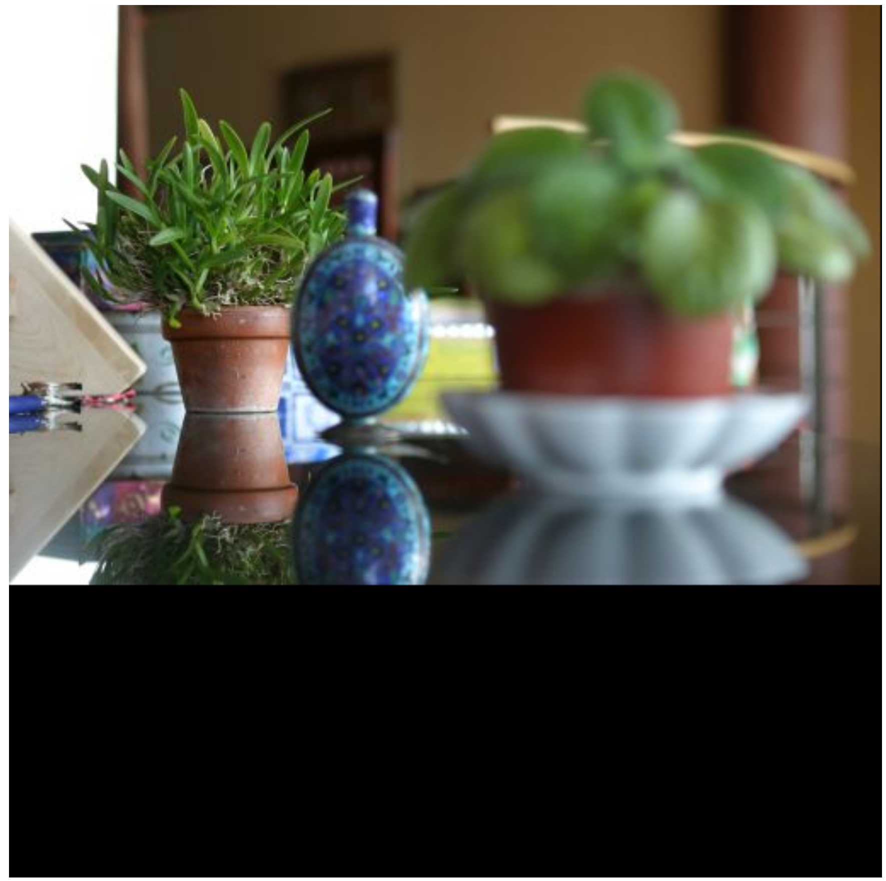
Input: Source images A, B & binary matte M.

Output: Blended image S

A



B



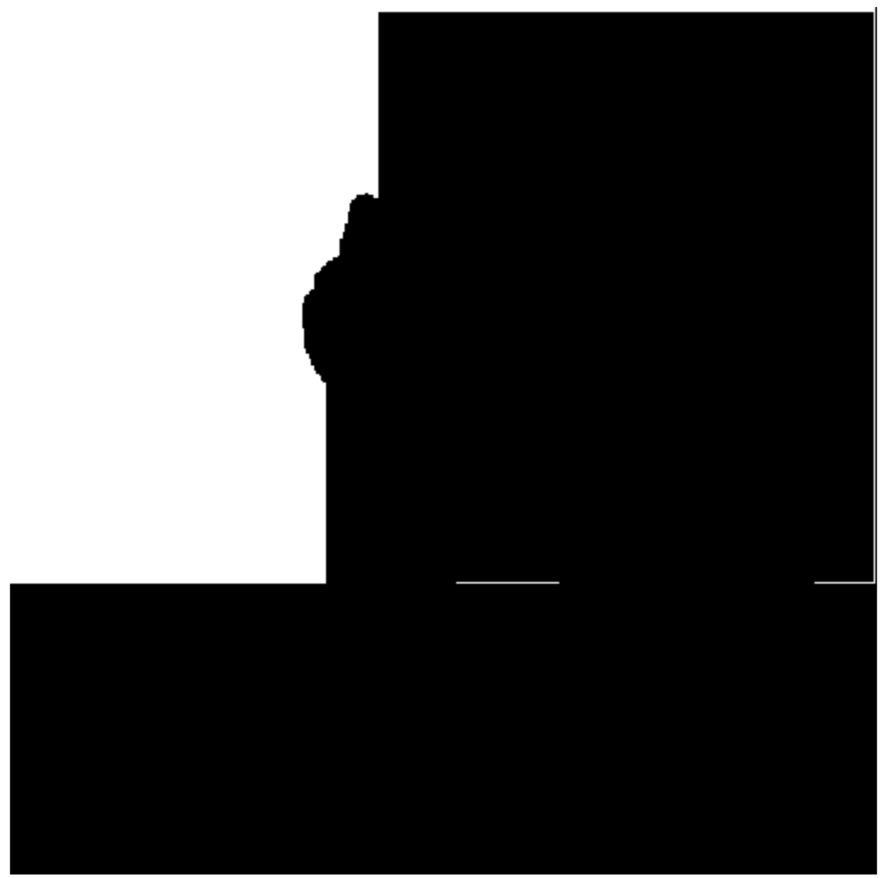
Pyramid Blending Algorithm

Input: Source images A, B & binary matte M.

Output: Blended image S

A

M



Pyramid Blending Algorithm

Input: Source images A, B & binary matte M

Output: Blended image S

① Compute A's Laplacian pyramid:

$AL_0, \ldots, AL_{N-1}, Ag_N$

Pyramid Blending Algorithm

Input: Source images A, B & binary matte M.
Output: Blended image S.

① Compute A's Laplacian pyramid: $AL_0, \ldots, AL_{N-1}, Ag_N$

② Compute B's Laplacian pyramid: $BL_0, \ldots, BL_{N-1}, Bg_N$

Pyramid Blending Algorithm

Input: Source images A, B & binary matte M.

Output: Blended image S

① Compute A's Laplacian pyramid: $AL_0, \ldots, AL_{N-1}, Ag_N$

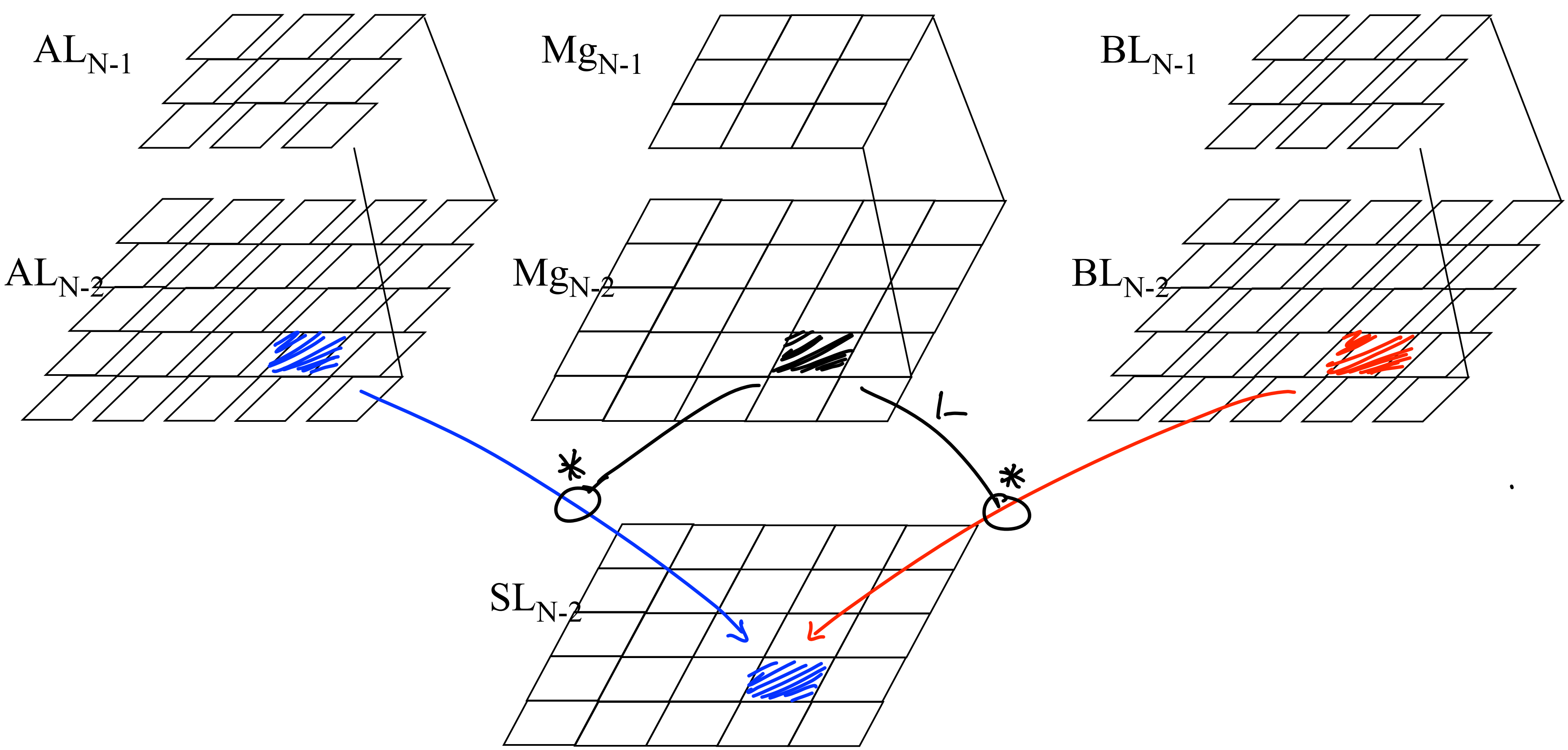
③ Compute M's Gaussian pyramid: $Mg_0, \ldots, Mg_N$

② Compute B's Laplacian pyramid: $BL_0, \ldots, BL_{N-1}, Bg_N$

Pyramid Blending Algorithm

(4) Compute the Laplacian pyramid, $SL_0, SL_1, \ldots SL_{N-1}, Sg_N$ by applying the matting equation with matte Mg_ℓ:

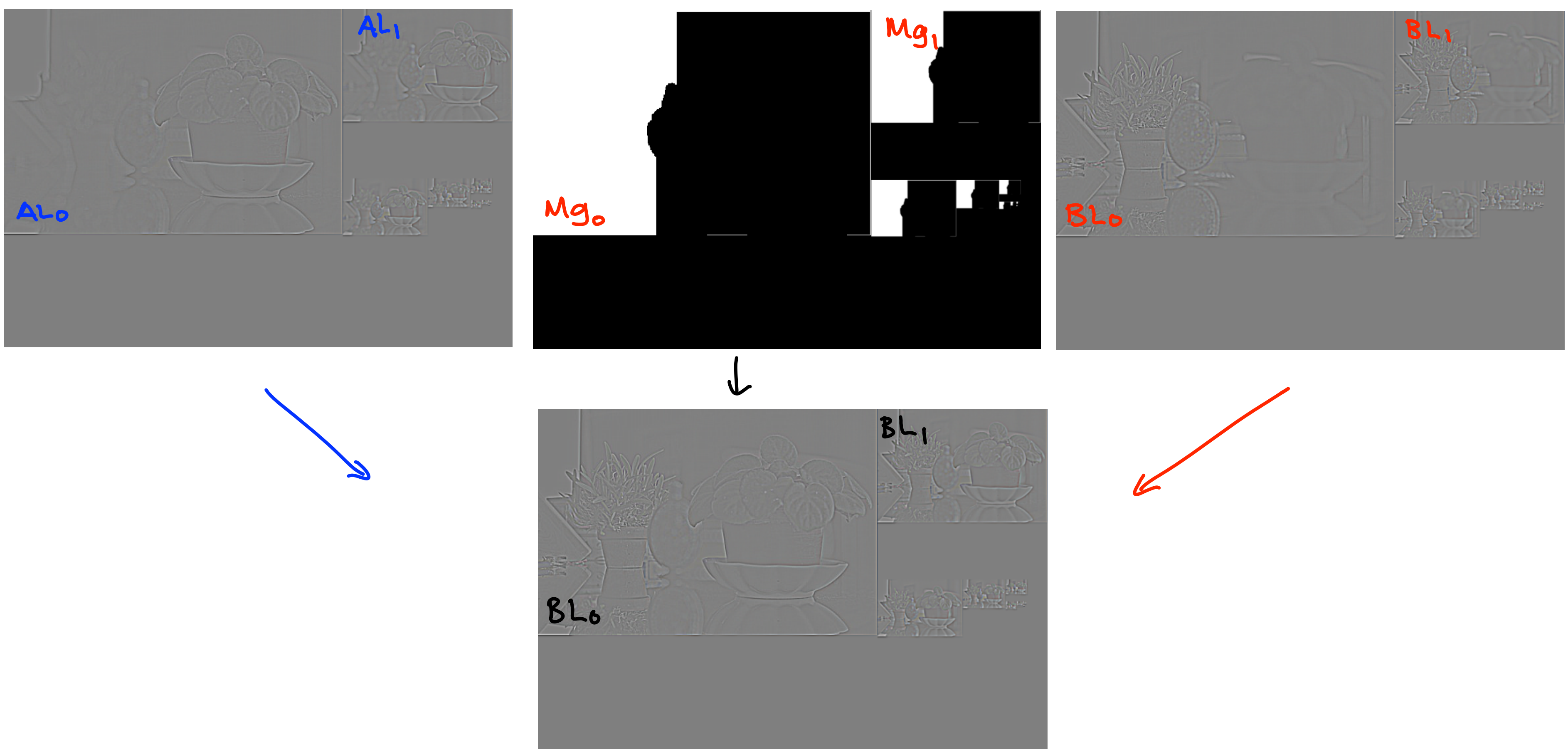
$$SL_\ell(i,j) = Mg_\ell(i,j)\,AL_\ell(i,j) + (1 - Mg_\ell(i,j))\,BL_\ell(i,j)$$



Pyramid Blending Algorithm

4 Compute the Laplacian pyramid, $SL_0, SL_1, \ldots SL_{N-1}, Sg_N$ by applying the matting equation with matte Mg_ℓ:

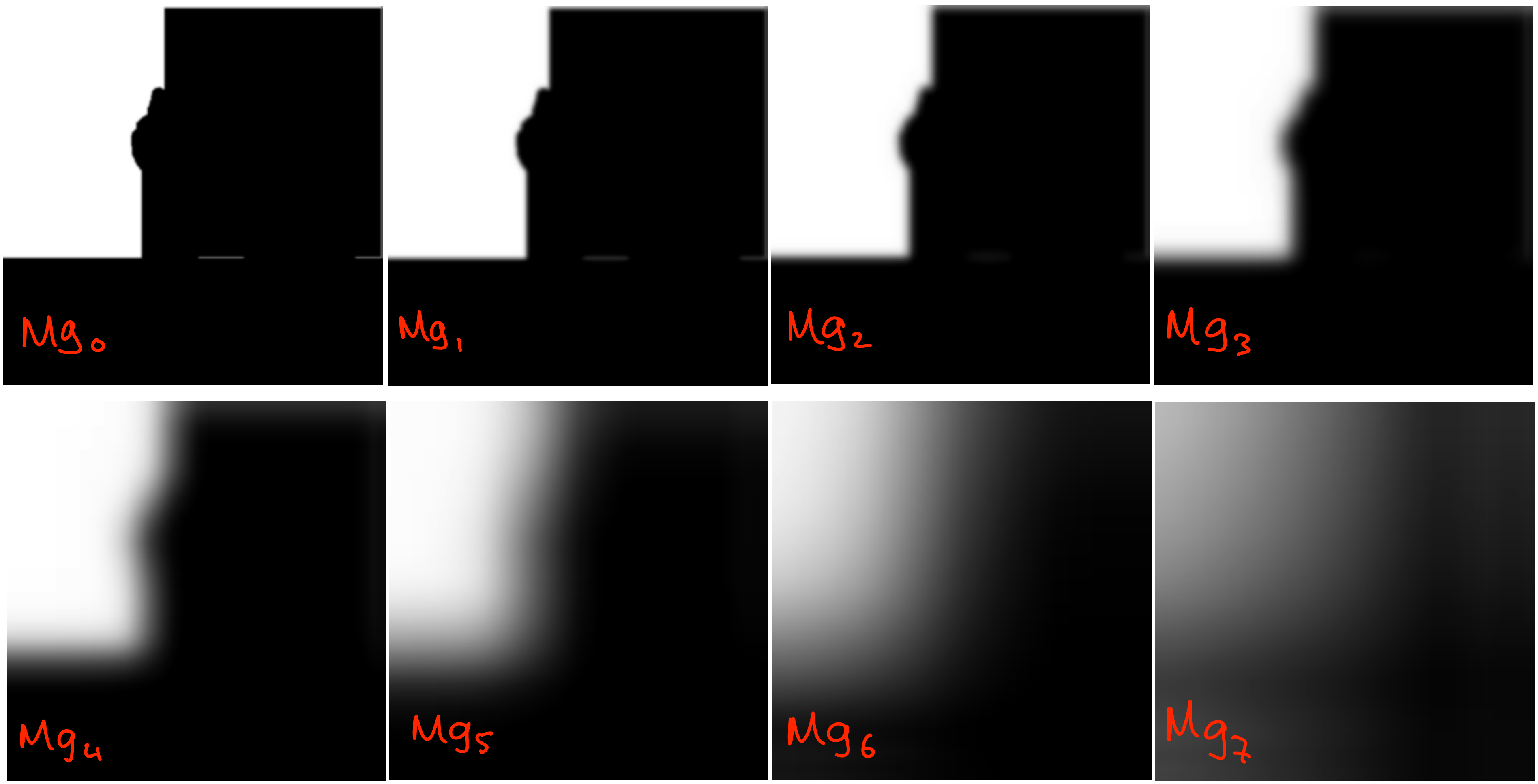
$$SL_\ell(i,j) = Mg_\ell(i,j)\, AL_\ell(i,j) + (1 - Mg_\ell(i,j))\, BL_\ell(i,j)$$



Pyramid Blending Algorithm

The algorithm effectively uses a different alpha matte for each level of detail

↑detail ⇒ ↓feathering window   ↓detail ⇒ ↑feathering window



Pyramid Blending Algorithm

(5) Compute level 0 of the Gaussian pyramid of S from $SL_0, \ldots, SL_{N-1}, Sg_N$

Result S'

$\Uparrow$

Pyramid Blending Algorithm

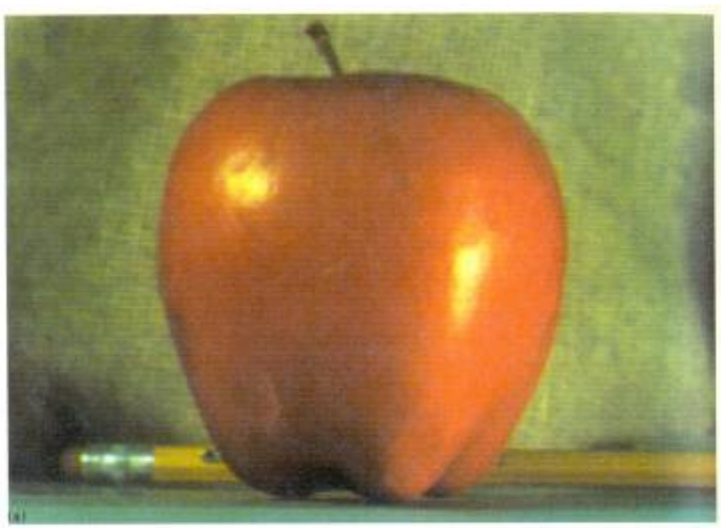
Blending Mis-Matched Photos (still looks OK)

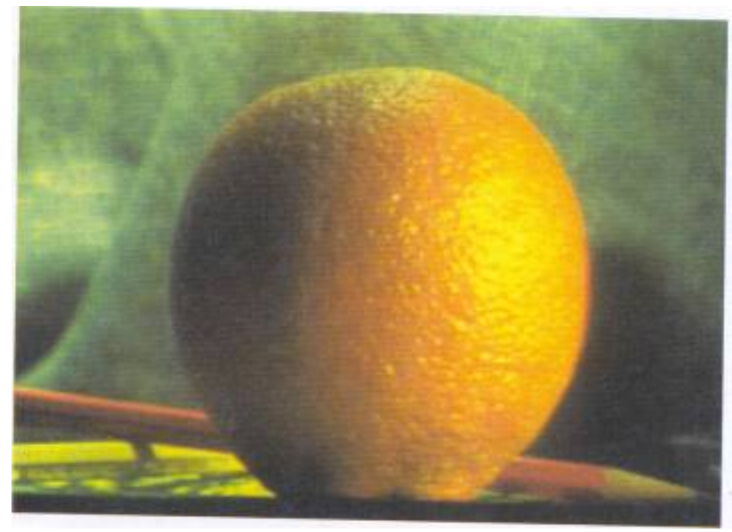
Merging Mis-Matched Photos (no blend)

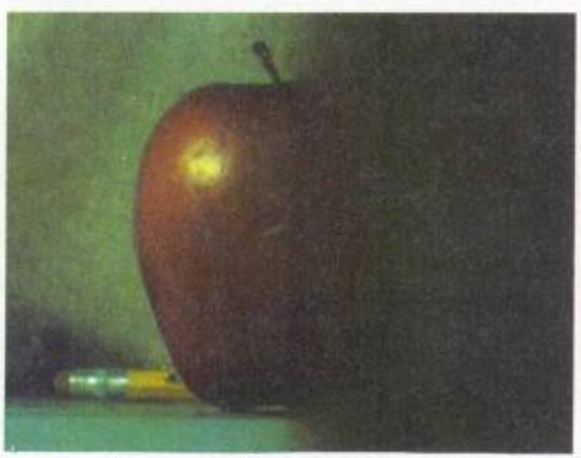
Blending without Using Pyramid

Blending without Using Pyramid

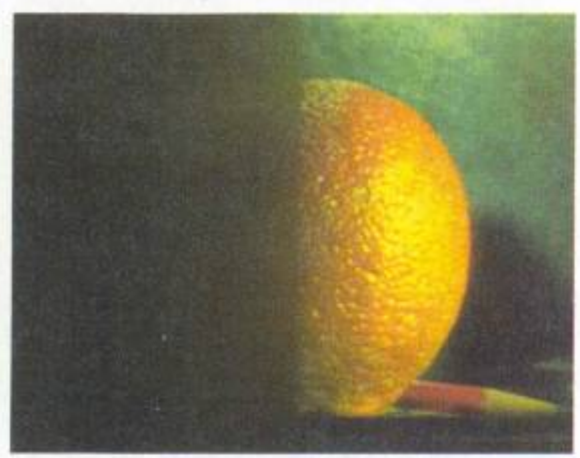
Pyramid Blending



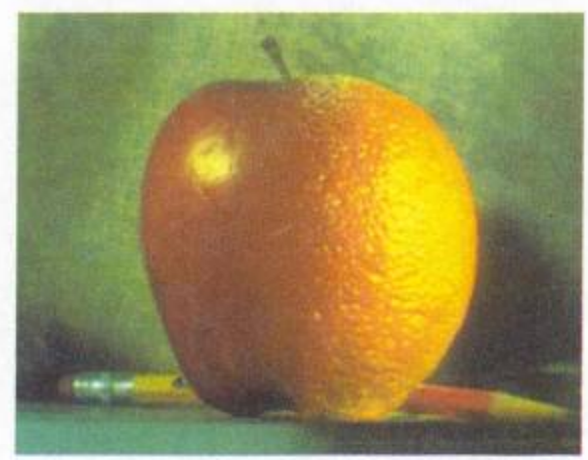




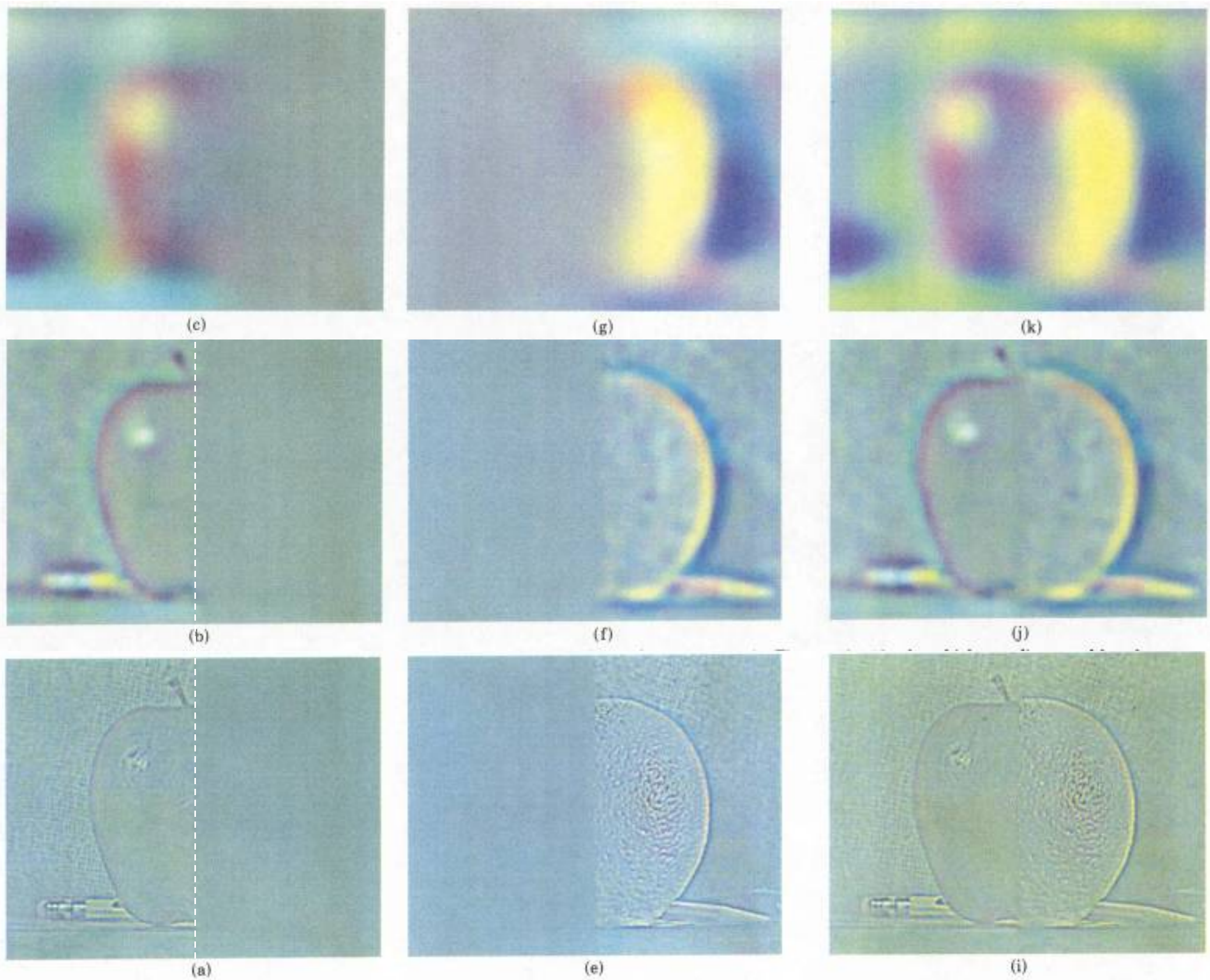
(d)



(h)



(l)



laplacian level 4

laplacian level 2

laplacian level 0

left pyramid

right pyramid

blended pyramid

Horror Photo